Bases de Données

Université Bordeaux 1

Alain Griffault

2010-2011

Outline

Outline

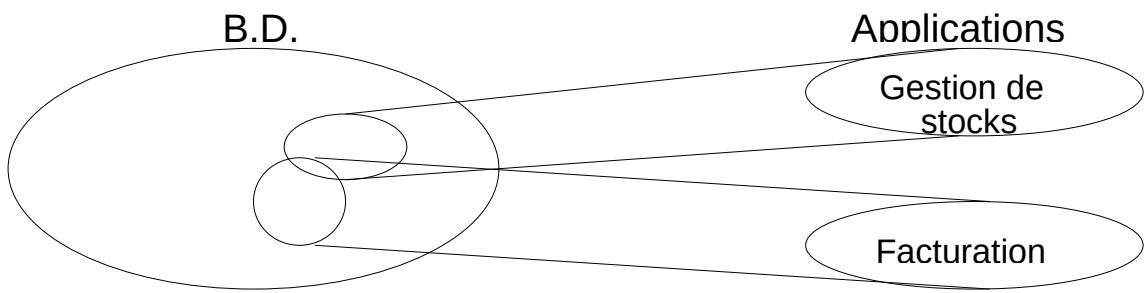
Historique

Objectifs

- Limiter le nombre d'interfaces utilisateurs.
- Diminuer les dépendances entre données et programmes.
- Diminuer les redondances, pour la place, pour les incohérences et les mises à jour.

Historique

Architecture d'un SGDB



Historique

Définitions

B.D. : Ensemble de données (sans redondance si possible) ou les liaisons entre les entités sont conservées; disponible simultanément pour plusieurs applications; et géré par un S.G.B.D.

S.G.B.D. : Ensemble de modules qui assurent l'interface entre les usagers et les informations. Outil qui permet la gestion d'informations (recherche, insertion, suppression, modification).

Historique

Architecture modulaire d'un SGBD

Ses caractéristiques (confidentialité, partage des ressources, accès concurrent et sécurité de fonctionnement) font qu'il est généralement partitionné fonctionnellement en trois parties:

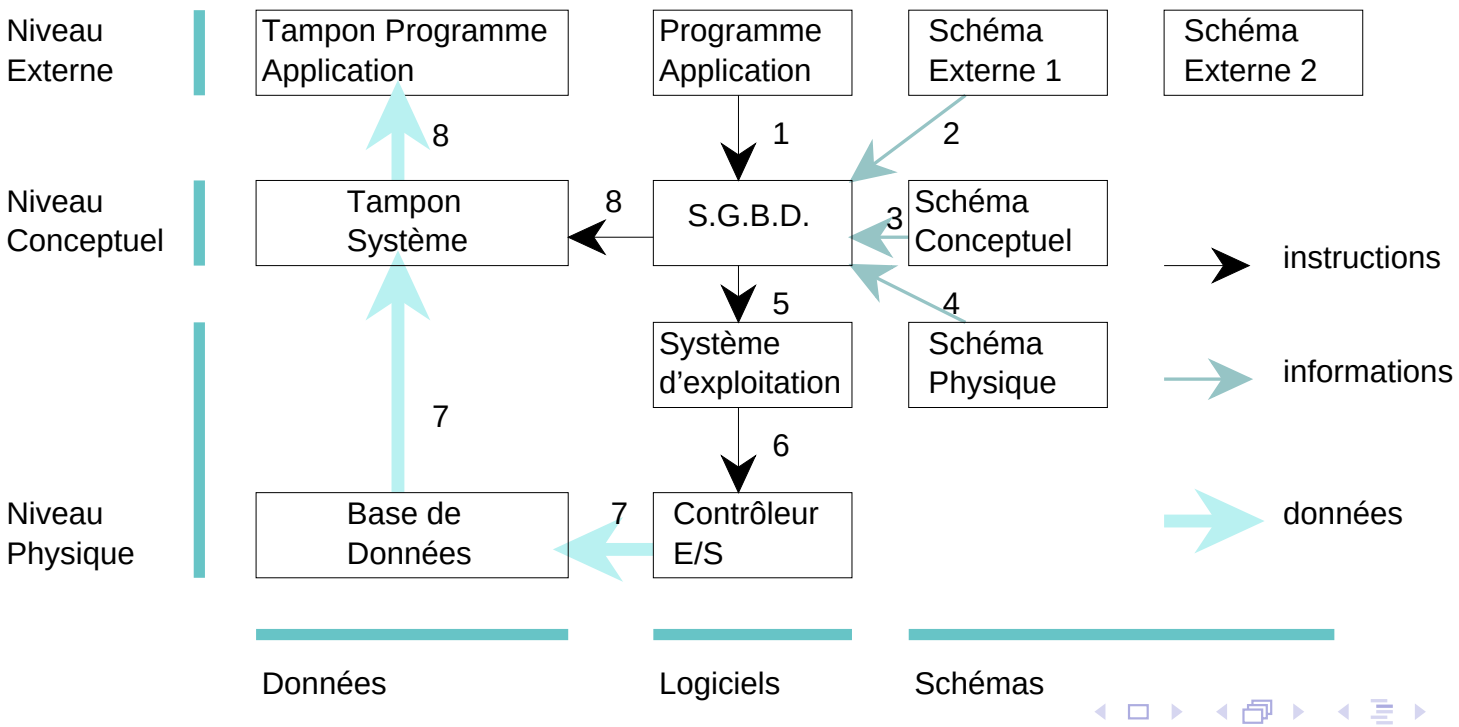
interne Les modules qui gèrent les liens entre les données manipulées, et les structures qui les stockent. Ils contiennent notamment toute l'algorithmique pour la gestion des informations.

externe Les modules qui produisent les images des données, qui sont utilisables par les programmes d'applications, et par les utilisateurs. Ils assurent les différents interfaces utilisateurs.

conceptuel/logique Les modules qui gèrent des fichiers d'informations propres au S.G.B.D. ainsi que des fichiers qui contiennent les schémas conceptuels des données.

Le traitement d'une requête

Les étapes du traitement d'une requête



Le traitement d'une requête

Exemple : l'âge d'une personne

Externe : Âge de Zinedine Zidane : 38 ans.

Conceptuel : Date de naissance de Zinedine : 23 juin 1972.

Physique : Plusieurs choix :

- trois entiers (23, 6, 1972).
- le nombre de jours entre naissance et une date (1,1,1900) : 26076.

Les problèmes

Le niveau physique

Le temps d'accès moyen à un disque (10 ms) est très supérieur au temps d'accès à la RAM. L'optimisation de la couche physique consiste donc à diminuer le nombre d'accès au disque. Les solutions concernent l'organisation et la compression des données.
Remarques :

- Pas de solution optimale pour tout type de problèmes;
- Un bon S.G.B.D. permet une grande variété de structures de stockage.
- Pour une grande base de données ($>>$ taille de la RAM), l'organisation des données est une tâche difficile, et dépend beaucoup de l'utilisation qui en est faîte.

Les problèmes

L'évolution

	Volume (octets)	Modèle
$\rightarrow$ 1985	Mega	Hiérarchique, Transactionnel, Gros serveur
1995	Giga	Relationnel, Client-serveur
2005	Tera	Entrepôt de données
...	Exa (10^{18})	Challenge

Le cours

Le contenu

Le cours traite des B.D relationnelles.

Étape 1 : nécessite un LMD. Un langage de :

- requêtes. OUI
- programmation incluant des requêtes. NON
- requêtes avec des structures de contrôles. NON

Étape 2 : nécessite un langage de description de liens entre les données applications et les données logiques. OUI

Étape 3 : nécessite un LDD. OUI

Étape 4 : nécessite un langage de description de liens entre données logiques et les structures de données. NON

Étape 5 : algorithmique optimisée pour demander au SE les pages contenant les données. OUI-NON

Étape 6 : gestionnaire de disques. NON

Le cours

Les contrôles

Un projet et un examen.

Bibliographie

- `http://lbdwww.epfl.ch/teaching/courses/poly1`
- *An Introduction to Database Systems, Eighth Edition* C.J. Date.

Polycopié et annales

- `http://dept-info.labri.fr/~griffaul/Enseignement/BD/`

Outline

Le modèle relationnel

Historique

- du à Codd en 1960. Nombreuses recherches jusque dans les années 1990. Modèle le plus utilisé par les SGBDs actuellement disponibles sur le marché.
- Modèle très simple. Trop simple pour certains $\rightarrow$ modèles conceptuels plus *riches* : modèles Entité-Association, relationnel-objet...

Un concept unique : la relation

Un exemple

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

Table: La relation étudiants de Bordeaux 1

Un concept unique : la relation

Notion de domaine

Un domaine D est un ensemble de valeurs atomiques (non décomposable).

- les chaînes de caractères de longueur maximale 30.
- les entiers positifs.
- les entiers compris entre 1 et 12.
- les couleurs $\{$Rouge, Vert, Bleu$\}$.
- ...

Un concept unique : la relation

Notion d'attribut

Un attribut A est une *caractéristique* d'un *objet* représenté par une valeur prise dans un domaine. On note $Dom(A)$ le domaine de l'attribut A.

Notion de n-uplet

Soit $\{A_1, A_2, \ldots, A_n\}$ une liste d'attributs. Un n-uplet ou tuple est un vecteur de valeurs
$(v_1 \in Dom(A_1), v_2 \in Dom(A_2), \ldots, v_n \in Dom(A_n))$.

Un concept unique : la relation

Notion de relation

Soit $\{A_1, A_2, \ldots, A_n\}$ une liste d'attributs. Une relation R est un ensemble de n-uplets
$\{(v_1 \in Dom(A_1), v_2 \in Dom(A_2), \ldots, v_n \in Dom(A_n))\}$.

Remarque : Une relation est un ensemble et non un multi-ensemble. Chaque n-uplet est donc unique, et il n'y a pas d'ordre entre les n-uplets.

Remarque : d'un point de vue ensembliste, $R \subseteq \prod_{i=1}^{i=n} Dom(A_i)$.

Ajout de sémantique dans le concept de relation

Identifiant ou clef

Soit une relation $R \subseteq \prod_{i=1}^{i=n} Dom(A_i)$.
L'ensemble $I = \{I_1, I_2, \ldots, I_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$ est un identifiant si et seulement si :

- il n'existe pas deux n-uplets dans R ayant même valeur sur l'ensemble I.
- pour tout sous-ensemble $J \subset I$, il existe deux n-uplets dans R ayant même valeur sur l'ensemble J.

Remarque : Pour la relation *Étudiants* de la table 1, le seul identifiant possible est *Numéro*.
Remarque : Si la relation R *évolue* au cours du temps, les ensembles identifiant peuvent évoluer aussi.
Remarque : Toute relation possède au moins une clef car la liste de tous les attributs vérifie le premier point.

Ajout de sémantique dans le concept de relation

Contrainte d'intégrité

Soit une relation $R \subseteq \prod_{i=1}^{i=n} Dom(A_i)$.
Soit P une propriété logique sur la liste d'attributs. P est une contrainte d'intégrité pour la relation R si et seulement si :

- P restreint à R est vrai.
- Toute modification de R laisse P vrai.

Ajout de sémantique dans le concept de relation

Types de contrainte d'intégrité

Élémentaire Pour chaque n-uplet v de la relation, il est possible de décider si $P(v)$ est vrai ou faux.

Statique Pour chaque n-uplet v de la relation, il n'est pas possible de décider si $P(v)$ est vrai ou faux, mais il est possible de décider si $P(R)$ est vrai ou faux.

Dynamique Il n'est pas possible de décider si $P(R)$ est vrai ou faux, mais il est possible de décider si $P(R \longrightarrow R)$ est vrai ou faux.

Remarque : Un identifiant est une contrainte d'intégrité statique particulière.

Ajout de sémantique dans le concept de relation

Schéma d'une relation

Schema(R) est constitué :

- d'un nom,
- d'une liste d'attributs $(A_1, A_2, \ldots, A_n)$ sur des domaines $(D_1, D_2, \ldots, D_n)$,
- d'un ensemble non vide d'identifiants (ou clefs) $\{I_i \subseteq \{A_1, A_2, \ldots, A_n\}\}$,
- d'une contrainte d'intégrité P (éventuellement *True*).

Remarque : par abus de language, on confond souvent le nom de la relation et le nom du schéma.

Un problème classique : une liste ordonnée de prénoms

Première solution

Numéro	Nom	Prénoms	Date de naissance
65758	nom1	prénom1 prénom3 prénom6	16 juin 1985
7897090	nom2	prénom2 prénom5	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1 prénom2	16 juin 1985
45365	nom3	prénom2 prénom4	20 septembre 1986
⋮	⋮	⋮	⋮

Défaut : Les attributs sont atomiques, les applications doivent faire l'analyse syntaxique de la liste des prénoms afin de n'afficher que le premier lorsque seul le premier est demandé.

Un problème classique : une liste ordonnée de prénoms

Deuxième solution

Numéro	Nom	Prénom1	Prénom2	Prénom3	Date de naissance
65758	nom1	prénom1	prénom3	prénom6	16 juin 1985
7897090	nom2	prénom2	prénom5		29 février 1984
7689689	nom1	prénom3			31 mars 1985
325	nom1	prénom1	prénom2		16 juin 1985
45365	nom3	prénom2	prénom4		20 septembre 1986
⋮	⋮	⋮	⋮	⋮	⋮

Défaut : Il faut fixer à priori le nombre maximal de prénoms $\rightarrow$ valeur *Non défini*.

Un problème classique : une liste ordonnée de prénoms

Troisième solution

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

Numéro	Ordre	Prénom
65758	2	prénom3
65758	3	prénom6
7897090	2	prénom5
325	2	prénom2
45365	2	prénom4
⋮	⋮	⋮

Un problème classique : une liste ordonnée de prénoms

Troisième solution : remarques

Cette description est satisfaisante si l'on peut spécifier que :

Clef {Numéro} est l'identifiant de la relation Étudiants

Clef {Numéro, Ordre} est l'identifiant de la relation AutresPrénoms

inter-relation $\forall n \in$ AutresPrénoms.Numéro, $\exists m \in$ Etudiants.Numéro tel que $n = m$

Elle serait peut-être mieux si l'on pouvait spécifier que :

Intégrité dynamique (num, ordre) $\in$ AutresPrénoms $\Rightarrow$ ordre $= 2$ ou (num, ordre-1) $\in$ AutresPrénoms.

Cette solution est sans doute la meilleure solution, car elle évite le problème de la valeur *Non défini*.

Le modèle conceptuel relationnel

Identifiant externe / Clef externe / Clef étrangère

Soient deux schémas de relations
$Schema(R^1) =< (A_1^1, A_2^1, \ldots, A_n^1), I^1, P^1 >$ et
$Schema(R^2) =< (A_1^2, A_2^2, \ldots, A_m^2), I^2, P^2 >$.
L'ensemble $I = \{I_1, I_2, \ldots, I_p\} \subseteq \{A_1^1, A_2^1, \ldots, A_n^1\}$ est un identifiant externe de la relation R^1 si et seulement si :

- I est un identifiant de R^2.
- $\forall (v_1^1, v_2^1, \ldots, v_n^1) \in R^1, \exists (v_1^2, v_2^2, \ldots, v_m^2) \in R^2$ tel que les valeurs $v_{I_1}, v_{I_2}, \ldots, v_{I_p}$ coincident.

Notion sémantique qui doit être déclarée par le concepteur. Le maintien de cette propriété sémantique est complexe.

Le modèle conceptuel relationnel

Contrainte d'intégrité inter-relations

Soit un ensemble de schémas de relations
$\{Schema(R^i) =< (A_1^i, A_2^i, \ldots, A_{n_i}^i), I^i, P^i >\}$.
Soit P un prédicat sur l'union des listes d'attributs. P est une contrainte d'intégrité inter-relations pour les relations R^i si et seulement si :

- P est vrai.
- Toute modification des relations R^i laisse P vrai.

Le modèle conceptuel relationnel

Schéma relationnel

- un ensemble de schémas de relations $\{Schema(R^i) =< (A^i_1, A^i_2, \ldots, A^i_{n_i}), \{I^i\}, P^i >\}$,
- pour chaque relation R^i, un ensemble (éventuellement vide) d'identifiants externes,
- une contrainte d'intégrité inter-relations (éventuellement *True*).

Outline

SQL : un langage de description de schémas relationnels

SQL (Structured Query Language)

- Le langage le plus utilisé aujourd'hui.
- Troisième version : SQL:92, SQL:99 et SQL:2003.

Description de schémas conceptuels

Création de tables

```
CREATE TABLE NomDeLaTable (
   A1 TypeDeDonnée
   ...
   AN TypeDeDonnée
);
```

Les types de données dépendent du SGBD. Classiquement, on trouve `CHAR(n)`, `INTEGER`, `FLOAT`, `DATE` ...

Description de schémas conceptuels

Déclaration de clés

Pour les clefs primaires, suivant si 1 attribut ou plusieurs :

```
PRIMARY KEY
PRIMARY KEY (A1, ..., AK)
```

et pour les clefs étrangères, suivant si 1 attribut ou plusieurs :

```
REFERENCES TableExterne(AI)
FOREIGN KEY (A1, ..., AK)
  REFERENCES TableExterne(AI1, ..., AIK)
```

Description de schémas conceptuels

Déclaration de contraintes

Plusieurs mots clefs : `ASSERTION`, `CHECK`, `CONSTRAINT`, `DEFAULT`, `NOT NULL`, `UNIQUE`, `TRIGGER`. Par exemple :

```
DateDeNaissance date CHECK (DateDeNaissance > 1/1/1
```

Les contraintes statiques et dynamiques peuvent être gérées soit par des fonctions SQL, soit par les `TRIGGER`, qui font des appels à des fonctions écrites en `C` ou dans un autre langage.

```
ALTER TABLE NomDeTable
  ADD CONSTRAINT NomContrainte CHECK condition;
```

```
CREATE TRIGGER NomDuTrigger
    BEFORE INSERT OR UPDATE OR DELETE ON NomTable
    FOR EACH ROW EXECUTE PROCEDURE trigf();
```

Description de schémas conceptuels

L'exemple des étudiants

```
CREATE TABLE Etudiants (
   Numero            integer   PRIMARY KEY,
   Nom               char(30)  NOT NULL,
   Prenom            char(30)  NOT NULL,
   DateDeNaissance   date      CHECK (DateDeNaissance > '1900

CREATE TABLE AutresPrenoms (
   Numero            integer   REFERENCES Etudiants(Numero)
   Ordre             integer   CHECK (Ordre > 1),
   Prenoms           char(30)  NOT NULL,
   PRIMARY KEY (Numero, Ordre),
   UNIQUE (Numero, Prenoms));
```

Modification du contenu d'une table

L'insertion, la modification et la suppression

```
INSERT INTO NomDeLaTable VALUES (v1, v2, ..., vn);
UPDATE NomDeLaTable SET (AI = vi, ... AJ=vJ)
   [WHERE condition];
DELETE FROM NomDeLaTable   [WHERE condition];
```

La table peut également être affectée lors de la création de la façon suivante :

```
CREATE TABLE NomDeLaTable (
   A1 TypeDeDonnée,
   ...
   AN TypeDeDonnée
) AS ''ResultatRequeteSelection'';
```

Modification du contenu d'une table

Exemple des étudiants

```
INSERT INTO Etudiants VALUES (45, 'DURAND', 'joe', '1987
INSERT INTO Etudiants VALUES (35, 'MARTIN', 'jack', '198
INSERT INTO Etudiants VALUES (70, 'DUPOND', 'averell', '
INSERT INTO Etudiants VALUES (70, 'DUPOND', 'averell', '
INSERT INTO Etudiants VALUES (55, 'DURAND', 'joe', '1956

INSERT INTO AutresPrenoms VALUES (45, 2, 'leo');
INSERT INTO AutresPrenoms VALUES (45, 3, 'ole');
INSERT INTO AutresPrenoms VALUES (35, 2, 'brice');
INSERT INTO AutresPrenoms VALUES (35, 3, 'jack');
```

Description de schémas externes

La vue

Une vue est une table virtuelle, qui permet une visualisation autre des données. La forme générale est :

```
CREATE VIEW NomDeLaVue (A1 ... AN)
   AS ``ResultatRequeteSelection'';
```

Description de schémas externes

Exemple des étudiants

```
CREATE VIEW ListeEtudiants (Nom, Prenom, Age) AS
  SELECT Nom, Prenom, EXTRACT(YEAR FROM current_date)
                      -EXTRACT(YEAR FROM DateDeNaissance
  FROM   Etudiants;
```

Outline

L'algèbre relationnelle

Ensemble d'opérations, qui à partir de relations, construisent d'autres relations.

Les opérateurs unaires sont la sélection (notée $\sigma[\,]$), la projection (notée $\pi[\,]$) et le renommage (noté $\alpha[\,]$).

Les opérateurs binaires sont l'union (noté $\cup$), la différence (notée $-$) et le produit (noté $\times$).

Les opérateurs déduits sont obtenus par composition des précédents. Ils sont pratiques, mais n'apportent aucune expressivité supplémentaire. On utilise généralement l'intersection (notée $\cap$), la jointure (notée $\bowtie$), la thêta-jointure (notée $\theta[\,]$) et la division (notée $/$).

Notations : Sauf indication contraire, la liste des attributs d'une relation R^i sera toujours $(A^i_1, A^i_2, \ldots, A^i_{n_i})$.

Les opérateurs unaires

La sélection (définition)

Definition

Soit une relation R.
Soit p un prédicat élémentaire sur les attributs $(A_1, A_2, \ldots, A_n)$.

$$\sigma[p](R) = \{(v_1, v_2, \ldots, v_n) \in R, \text{ et } p(v_1, v_2, \ldots, v_n)\}$$

La sélection construit la relation constituée des n-uplets de R qui satisfont p. (La sélection supprime des n-uplets).

Les opérateurs unaires

La sélection (exemple)

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

$\rightarrow$

Numéro	Nom	Prénom	Date de naissance
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
⋮	⋮	⋮	⋮

Table: L'opération
$\sigma[(\texttt{Nom} = \text{“}\texttt{nom2}\text{”})|(\texttt{Datedenaissance} = \text{“}\texttt{31 mars 1985}\text{”})](\texttt{Etudiants})$

Les opérateurs unaires

La sélection (complexité)

Dans le pire des cas, il faut itérer sur tous les n-uplets de la relation.

$$O(|R|)$$

Les opérateurs unaires

La projection (définition)

Definition

Soit une relation R.
Soit $S = \{S_1, \ldots, S_m\} \subset A$ un sous-ensemble des attributs. Posons $B = A - S = \{B_1, \ldots, B_{n-m}\}$.
$\pi[S](R) =$
$\{(v_1, \ldots, v_m) / \exists (b_1, \ldots, b_{n-m})\}$ et $(v_1, \ldots, v_m, b_1, \ldots, b_{n-m}) \in R\}$

La projection construit la relation obtenue à partir de R en éliminant les attributs non présents dans S.

Les opérateurs unaires

La projection (exemple)

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
325	nom1	prénom1	21 juin 1987
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

$\rightarrow$

Nom	Prénom
nom1	prénom1
nom2	prénom2
nom1	prénom1
nom3	prénom2
⋮	⋮

$\rightarrow$

Nom	Prénom
nom1	prénom1
nom2	prénom2
nom3	prénom2
⋮	⋮

Table: L'opération $\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$

Les opérateurs unaires

La projection (complexité)

Dans le pire des cas, il faut itérer sur tous les n-uplets de la relation, puis enlever les doublons éventuels ce qui nécessite un tri.

$$O(|R|log_2(|R|))$$

Si la projection contient l'identifiant, il n'y a pas de doublon et la complexité est :

$$O(|R|)$$

Les opérateurs unaires

Le renommage (définition)

Definition

Soit une relation R.
Soit B_i un attribut ayant même domaine que l'attribut A_i.
$\alpha[A_i : B_i](R) = \{(v_1, v_2, \ldots, v_n) \in R\}$

Le renommage construit la relation obtenue à partir de R en remplaçant certains attributs.

Les opérateurs unaires

Le renommage (exemple)

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
325	nom1	prénom1	21 juin 1987
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

$\rightarrow$

Numéro	Nom	Prénom	Né(e) le
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
325	nom1	prénom1	21 juin 1987
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

Table: L'opération $\alpha[(\texttt{Datedenaissance} : \texttt{Ne(e)le})](\texttt{Etudiants})$

Les opérateurs unaires

Le renommage (complexité)

$$\Theta(1)$$

Les opérateurs binaires

L'union (définition)

Definition

Soient R^1 et R^2 deux relations ayant la même liste d'attributs.
$R^1 \cup R^2 = \{v = (v_1, \ldots, v_n)$ `tel` que $v \in R^1$ ou $v \in R^2\}$

L'union construit la relation constituée des n-uplets appartenant soit à R^1 soit à R^2.

Les opérateurs binaires

L'union (exemple)

lettre	chiffre
A	7
F	6
M	7

$\cup$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
F	6
M	7
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
F	6
M	7
C	4
Z	9

Table: L'union de deux relations

Les opérateurs binaires

L'union (complexité)

Il faut enlever les doublons. Dans le pire des cas, il faut pour chacun des tuples d'une relation, tester son existence dans l'autre relation pour savoir s'il faut ajouter ou non le tuple.
Il faut donc trier la plus petite des deux, puis utiliser un algorithme de recherche dichotomique.

$$O((|R^1| + |R^2|)log_2(|R^i|)$$

avec

$$|R^i| = min(|R^1|, |R^2|)$$

Les opérateurs binaires

La différence (définition)

Definition

Soient R^1 et R^2 deux relations ayant la même liste d'attributs.
$R^1 - R^2 = \{v = (v_1, \ldots, v_n)$ `tel` que $v \in R^1$ `et` $v \notin R^2\}$

La différence construit la relation constituée des n-uplets appartenant à R^1 et n'appartenant pas à R^2.

Les opérateurs binaires

La différence (exemple)

lettre	chiffre
A	7
F	6
M	7
C	4

$-$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
M	7

Table: La différence de deux relations

Les opérateurs binaires

La différence (complexité)

Deux possibilités : Pour chaque n-uplet de R^2, il faut l'enlever de R^1, ou bien pour chaque n-uplet de R^1, il faut tester son existence dans R^2. Dans le pire des cas, chacune des possibilités demandent de trier une relation.
Il faut donc trier la plus petite des deux, puis utiliser un algorithme de recherche dichotomique.

$$O(|R^i|log_2(|R^i| + |R^j|log_2(|R^i|)) = O((|R^1| + |R^2|)log_2(|R^i|)$$

avec

$$|R^i| = min(|R^1|, |R^2|)$$

Les opérateurs binaires

Le produit cartésien (définition)

Definition

Soient R^1 et R^2 deux relations telles que
$\{A_1^1, \ldots, A_{n_1}^1\} \cap \{A_1^2, \ldots, A_{n_2}^2\} = \emptyset$
$R^1 \times R^2 = \{(v_1^1, \ldots, v_{n_1}^1, v_1^2, \ldots, v_{n_2}^2)$ tel que $(v_1^1, \ldots, v_{n_1}^1) \in R^1$ et $(v_1^2, \ldots, v_{n_2}^2) \in R^2\}$

Le produit construit la relation constituée des n-uplets obtenus en combinant tous les n-uplets de R^1 à tous ceux de R^2.

Les opérateurs binaires

Le produit cartésien (exemple)

lettre	chiffre
A	7
F	6
C	4

$\times$

indice	pays
33	F
39	I

$=$

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	33	F
F	6	39	I
C	4	33	F
C	4	39	I

Table: Le produit cartésien de deux relations

Les opérateurs binaires

Le produit cartésien (complexité)

Pour chaque n-uplet de R^2, il faut itérer R^1.

$$O(|R^1| \times |R^2|)$$

Les opérateurs déduits

L'intersection (définition)

Definition

Soient R^1 et R^2 deux relations ayant la même liste d'attributs.
$R^1 \cup R^2 = \{v = (v_1, \ldots, v_n)$ `tel` que $v \in R^1$ ou $v \in R^2\}$

L'intersection construit la relation constituée des n-uplets appartenant à R^1 et à R^2.

Les opérateurs déduits

L'intersection (exemple)

lettre	chiffre
A	7
F	6
M	7

$\cap$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
F	6

Table: L'intersection de deux relations

Les opérateurs déduits

L'intersection (remarques)

Propriété

Soient R^1 et R^2 deux relations ayant la même liste d'attributs.
$R^1 \cap R^2 = R^1 - (R^1 - R^2) = R^2 - (R^2 - R^1)$

Complexité : Celle de la différence.

$$O(|R^i| log_2(|R^i| + |R^j| log_2(|R^i|)) = O((|R^1| + |R^2|) log_2(|R^i|)$$

avec

$$|R^i| = min(|R^1|, |R^2|)$$

Les opérateurs déduits

La thêta-jointure (définition)

Definition

Soient R^1 et R^2 deux relations ayant des ensembles d'attributs disjoints.
Soit P un prédicat élémentaire sur l'union des attributs.
$R^1 \theta[P] R^2 = \sigma[P](R^1 \times R^2)$

La thêta-jointure construit le produit et ne conserve que les n-uplets qui vérifient P.

Les opérateurs déduits

La thêta-jointure (exemple)

lettre	chiffre
A	7
F	6
C	4

$\theta[\texttt{lettre} \neq \texttt{pays}]$

indice	pays
33	F
39	I

$=$

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	39	I
C	4	33	F
C	4	39	I

Table: La thêta-jointure de deux relations

Les opérateurs déduits

La thêta-jointure (complexité)

Celle du produit.

$$O(|R^1| \times |R^2|)$$

Les opérateurs déduits

La jointure naturelle (définition)

Definition (Jointure)

La jointure naturelle construit la relation constituée des n-uplets obtenus en combinant tous les n-uplets de R^1 à tous ceux de R^2 ayant les mêmes valeurs sur les attributs communs.

Les opérateurs déduits

La jointure naturelle (exemple)

lettre	chiffre
A	7
F	6
C	4

$\bowtie$

indice	lettre
33	F
7	F
39	I

$=$

lettre	chiffre	indice
F	6	33
F	6	7

Table: La jointure naturelle de deux relations

Les opérateurs déduits

La jointure naturelle (remarques)

Un renommage pour éviter la confusion, puis un produit, puis la sélection et enfin la projection pour enlever l'attribut renommé devenu inutile car redondant.

Propriété

Soient R^1 *une relation sur les attributs* $(A_1, \ldots, A_n, B_1^1, \ldots, B_{n_1}^1)$ *et* R^2 *une relation sur* $(A_1, \ldots, A_n, B_1^2, \ldots, B_{n_2}^2)$.
$R^1 \bowtie R^2 = \pi[A_i, B_j^1, B_k^2](\sigma[A_i = R^2.A_i](R^1 \times (\alpha[A_i : R^2.A_i](R^2))))$

Complexité : Celle du produit car par construction il n'y a pas de doublon.

$$O(|R^1| \times |R^2|)$$

Les opérateurs déduits

La division (définition)

Definition

La division de R^1 par R^2 construit la plus grande relation dont le produit avec R^2 est inclus dans R^1.

Remarque : Cela ressemble à une division entière.

Les opérateurs déduits

La division (exemple)

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	39	I
C	4	33	F
C	4	39	I

/

indice	pays
33	F
39	I

=

lettre	chiffre
A	7
C	4

Table: La division de deux relations

Les opérateurs déduits

La division (complexité)

On projette R^1 sur ses attributs propres, puis le produit avec R^2 fourni un sur-ensemble de R^1. La différence avec R^1 représente les n-uplets *non entiers* de R^1, ce sont donc ceux-là qu'ils faut retrancher de la projection de R^1.

Propriété

Soient R^1 une relation sur les attributs $(A_1, \ldots, A_n, B_1, \ldots, B_n)$ et R^2 une relation sur $(B_1, \ldots, B_n)$.
$R^1 / \ R^2 = \pi[A_i](R^1) - \pi[A_i]((\pi[A_i](R^1) \times R^2) - R^1)$

Complexité : *vue en TD*

Quelques propriétés de l'algèbre relationnelle

Les opérateurs commutatifs et associatifs

$\forall$ op $\in \{\cup, \cap, \times, \bowtie, \theta[p]\}$,

- R^1 op $R^2 = R^2$ op R^1
- R^1 op $(R^2$ op $R^3) = (R^1$ op $R^2)$ op R^3.

Quelques propriétés de l'algèbre relationnelle

Les compositions d'opérateurs

- $\pi[A_i^1](\pi[A_i^2](\ldots(\pi[A_i^k](R)))) = \pi[\cap_{j=1}^{j=k} A_i^j](R)$
- $\sigma[P^1](\sigma[P^2](\ldots(\sigma[P^k](R)))) = \sigma[\bigwedge_{j=1}^{j=k} P^j](R)$
- $(R^1 \times R^2)/\ R^2 = R^1$
- $(R^1/\ R^2) \times R^2 \subseteq R^1$

Quelques propriétés de l'algèbre relationnelle

Remarques

- La différence est ni commutative, ni associative.
- Le renommage se compose avec *délicatesse*.
- L'opération de complément n'est pas définie.

Quelques propriétés de l'algèbre relationnelle

Optimisation des calculs

En utilisant les propriétés des opérateurs et leurs complexités, il est possible par simple réécriture des calculs, d'optimiser le temps nécessaire à l'obtention du résultat.
En réalité, le processus d'optimisation des requêtes est beaucoup plus complexe, car il tient compte également des structures de données utilisées et des tailles des différentes relations.

Outline

SQL : un langage algébrique de manipulation de données

SQL (Structured Query Language)

- SQL est le langage de requêtes le plus utilisé aujourd'hui par les SGBD Relationnels.
- SQL permet d'exprimer tous les opérateurs de l'algèbre relationnelle.
- Les exemples donnés respectent la syntaxe SQL du SGBD `PostgreSQL 8.0.3`
- Pour un autre SGBD, la syntaxe peut varier légèrement, et même beaucoup si les opérateurs implémentés diffèrent.
- **Attention :** Les SGBD implémentent souvent les relations par des multi-ensembles et non des ensembles. Les doublons doivent être gérés explicitement.

SQL : un langage algébrique de manipulation de données

Forme générale d'une requête SQL

`SELECT` *Liste des noms d'attributs du résultat*
`FROM` *Liste de relations contenant les attributs précédents*
`[WHERE` *un prédicat sur l'union des attributs des relations précédentes* `] ;`

Les opérateurs unaires

La sélection (syntaxe)

Soit R une relation sur les attributs $(A_1, A_2, \ldots, A_n)$.
Soit p un prédicat élémentaire sur les attributs $(A_1, A_2, \ldots, A_n)$.
Le calcul de $\sigma[p](R)$ s'écrit en SQL :

```
SELECT A1, A2, ..., An
FROM   R
WHERE  p;
```

ou plus simplement

```
SELECT *  /* l'etoile signifie tous les attributs */
FROM   R
WHERE  p;
```

Les opérateurs unaires

La sélection (exemple)

$$\sigma[(\texttt{Nom} = \text{“}\texttt{MARTIN}\text{”})|(\texttt{DateDeNaissance} = \text{“}\texttt{13juin1987}\text{”})](\texttt{Etudiants})$$

```
SELECT *
FROM   Etudiants
WHERE  (Nom='MARTIN')
    OR (DateDeNaissance='1987-06-13') ;
```

Les opérateurs unaires

La projection (syntaxe)

Soit R une relation sur les attributs $A = \{A_1, A_2, \ldots, A_n\}$.
Soit $S = \{S_1, S_2, \ldots, S_m\} \subset A$ un sous-ensemble des attributs.
Le calcul de $\pi[S](R)$ ne s'écrit pas en SQL :

```
SELECT S1, S2, ..., Sn
FROM R;
```

mais

```
SELECT DISTINCT S1, S2, ..., Sn
FROM R;
```

car la norme SQL n'indique pas toujours la suppression des doublons dans les résultats des requêtes.

Les opérateurs unaires

La projection (exemple)

$$\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$$

```
SELECT DISTINCT Nom, Prenom
FROM   Etudiants;
```

Les opérateurs unaires

Le renommage explicite des attributs

```
SELECT Attribut1 AS XXX, ..., AttributN
FROM   R;
```

également possible dans la définition des vues.

```
CREATE VIEW NomDeLaVue (NomAttribut1 ... NomAttributN) AS
SELECT Attribut1, ..., AttributN
FROM   R;
```

Les opérateurs unaires

Le renommage implicite des attributs

Utilisatiion de la notation pointée.

```
SELECT R.NomAttributI
FROM   R;
```

qui est équivalent à :

```
SELECT NomAttributI
FROM   R;
```

Les opérateurs unaires

Le renommage explicite des relations

```
SELECT NomAttributI
FROM   R RR
WHERE  RR.NomAttributK='Dupont';
```

Les opérateurs binaires

L'union (syntaxe)

Soient R^1, R^2 deux relations sur les mêmes attributs. Le calcul de $R^1 \cup R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1
UNION
SELECT *
FROM   R2;
```

Par défaut, les doublons sont éliminés. Si l'on veut les conserver :

```
SELECT *
FROM   R1
UNION ALL
SELECT *
FROM   R2;
```

Les opérateurs binaires

L'union (exemple)

Tous les prénoms pour tous les étudiants.

$\pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{Etudiants}) \cup \pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{AutresPrenoms})$

```
SELECT Numero, Prenom
FROM   Etudiants
UNION
SELECT Numero, Prenom
FROM   AutresPrenoms;
```

Les opérateurs binaires

La différence (syntaxe)

Soient R^1, R^2 deux relations sur les mêmes attributs. Le calcul de $R^1 - R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1

EXCEPT /* ou bien MINUS */

SELECT *
FROM   R2;
```

Les opérateurs binaires

La différence (exemple)

Tous les numéros d'étudiants ayant un seul prénom.

$$\pi[(\texttt{Numero})](\texttt{Etudiants}) - \pi[(\texttt{Numero})](\texttt{AutresPrenoms})$$

```
SELECT Numero
FROM   Etudiants
EXCEPT
SELECT Numero
FROM   AutresPrenoms;
```

Les opérateurs binaires

Le produit cartésien (syntaxe)

Soient R^1, R^2 deux relations *non nécessairement sur des ensembles attributs disjoints*.
Le calcul de $R^1 \times R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1, R2;
```

Les requêtes imbriquées

La composition (syntaxe)

La composition des opérateurs de l'algèbre s'écrit aisément.

```
SELECT liste1Attributs
FROM   R1,
       (SELECT liste2Attributs
        FROM RR1,
             (SELECT liste3Attributs
              FROM RRR1, RRR2, ... RRRn) AS RR2,
             RR3, ... RRn) AS R2,
       R3, ... Rn    ;
```

Les opérateurs déduits

L'intersection (syntaxe)

Soient R^1, R^2 deux relations sur les mêmes attributs.
Le calcul de $R^1 \cap R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1

INTERSECT

SELECT *
FROM   R2;
```

Les opérateurs déduits

L'intersection (alternative)

L'intersection n'est pas toujours un opérateur implémenté. Il suffit alors de revenir à l'algèbre relationnelle :
$R^1 \cap R^2 = R^1 - (R^1 - R^2)$

```
SELECT *
FROM   R1
EXCEPT (
  SELECT *
  FROM   R1
  EXCEPT
  SELECT *
  FROM   R2
);
```

Les opérateurs déduits

L'intersection (exemple)

Des erreurs de saisie sur les prénoms (prénoms en double).

$\pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{Etudiants}) \cap \pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{AutresPrenoms})$

```
SELECT Numero, Prenom
FROM   Etudiants
INTERSECT
SELECT Numero, Prenom
FROM   AutresPrenoms;
```

Les opérateurs déduits

La thêta-jointure (syntaxe)

Soient R^1, R^2 deux relations sur des ensembles d'attributs disjoints.
Soit P un prédicat élémentaire sur l'union des attributs.
Le calcul de $R^1 \theta[P] R^2$ s'écrit en SQL

```
SELECT *
FROM   R1, R2
WHERE  P;
```

C'est la syntaxe de base d'une requête SQL.

Les opérateurs déduits

La thêta-jointure (exemple)

Les étudiants ayant le même premier prénom.

$$\texttt{Etudiants}\ \theta[\mathit{Prenom} = \mathit{Prenom}]\ \texttt{Etudiants}$$

```
SELECT *
FROM   Etudiants E1, Etudiants E2
WHERE  (E1.prenom = E2.prenom)
   AND (E1.Numero < E2.Numero);
```

Les opérateurs déduits

La jointure naturelle (syntaxe)

Soient R^1, R^2 deux relations sur les attributs $(A_1, \ldots, A_n, B_1, \ldots, B_m)$ et $(A_1, \ldots, A_n, C_1, \ldots, C_p)$.
Le calcul de $R^1 \bowtie R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1 NATURAL JOIN R2;
```

Les opérateurs déduits

La jointure naturelle (alternative)

La jointure naturelle est presque toujours implémenté. Sinon, il suffit de revenir à l'algèbre relationnelle :

```
SELECT R1.A1, ..., R1.An, R1.B1, ..., R1.Bm,
       R2.C1, ...,R2.Cp
FROM   R1, R2
WHERE  R1.A1 = R2.A1
   AND R1.A2 = R2.A2
   AND ...
   AND R1.An = R2.An;
```

Remarque : Les SGBD distinguent quelques fois les `NATURAL JOIN`, les `INNER JOIN`, les `OUTER JOIN` et les `CROSS JOIN`. (cf doc)

Les opérateurs déduits

La jointure naturelle (exemple)

$$\texttt{Etudiants} \bowtie \texttt{AutresPrenoms}$$

```
SELECT *
FROM   Etudiants NATURAL JOIN AutresPrenoms;

SELECT Etudiants.Numero, Nom,
       Etudiants.Prenom, DateDeNaissance, Ordre
FROM   Etudiants, AutresPrenoms
WHERE  Etudiants.Numero = AutresPrenoms.Numero;
```

Les opérateurs déduits

La division (syntaxe)

$R^1 : (A_1, \ldots, A_n, B_1, \ldots, B_n)$ et $R^2 : (B_1, \ldots, B_n)$.
Remarque : très rarement implémenté, donc :

$$R^1 / R^2 = \pi[A_i](R^1) - \pi[A_i]((\pi[A_i](R^1) \times R^2) - R^1)$$

```
SELECT DISTINCT A1, A2, ..., An
FROM   R1
MINUS
SELECT A1, A2, ..., An
FROM   (SELECT *
        FROM   (SELECT DISTINCT A1, A2, ..., An,
                FROM   R1) AS Alias1,
               R2
        MINUS
        SELECT *
        FROM   R1) AS Alias2;
```

Les opérateurs déduits

La division (exemple) (1)

```
CREATE TABLE R1 (
   lettre   char(1),
   chiffre  integer,
   indice   integer,
   pays     char(1),
PRIMARY KEY (lettre, chiffre, indice, pays)
);

INSERT INTO R1 VALUES ('A', 7, 33, 'F');
INSERT INTO R1 VALUES ('A', 7, 39, 'I');
INSERT INTO R1 VALUES ('A', 7, 34, 'G');
INSERT INTO R1 VALUES ('F', 6, 39, 'I');
INSERT INTO R1 VALUES ('C', 4, 33, 'F');
INSERT INTO R1 VALUES ('C', 4, 39, 'I');
```

Les opérateurs déduits

La division (exemple) (2)

```
CREATE TABLE R2 (
   indice   integer,
   pays     char(1),
PRIMARY KEY (indice, pays)
);

INSERT INTO R2 VALUES (33, 'F');
INSERT INTO R2 VALUES (39, 'I');
```

Les opérateurs déduits

La division (exemple) (3)

```
-- les couples (lettre, chiffre) pour lesquels
-- tous les tuples de R2 forment des tuples de R1
SELECT DISTINCT lettre, chiffre
FROM   R1
EXCEPT
SELECT lettre, chiffre
FROM   (SELECT *
        FROM   (SELECT DISTINCT lettre, chiffre
                FROM   R1) AS PiR1,
-- Alias obligatoire
                R2
        EXCEPT
        SELECT *
        FROM   R1) AS NonEntierR1;
-- Alias obligatoire
```

Outline

Théorie de la normalisation

Objectifs

La théorie de la normalisation a pour objectif de pouvoir comparer sur différents critères plusieurs schémas conceptuels contenant les mêmes informations.

Les critères

- Espace requis pour le stockage.
- Espace requis pour le traitement.
- Temps mis pour les traitements des données.
  - Temps mis pour l'insertion de nouvelles données.
  - Temps mis pour la suppression de données.
  - Temps mis pour la modification de données.
  - Temps mis pour l'accès aux données.

Théorie de la normalisation

Résultats

La théorie de la normalisation ne permet pas de dire qu'une solution est optimale, mais elle permet d'expliquer les défauts de certaines solutions.

Quel schéma conceptuel choisir (exemple) ?

Premier schéma

```
CREATE TABLE R (
   A   char(2),
   B   char(2),
   C   char(2),
PRIMARY KEY (A, B)
);
-- Une contrainte pour verifier ((ai = aj) => (ci =

INSERT INTO R VALUES ('a1', 'b1', 'c2');
INSERT INTO R VALUES ('a1', 'b2', 'c2');
INSERT INTO R VALUES ('a2', 'b1', 'c1');
INSERT INTO R VALUES ('a3', 'b2', 'c1');
```

Une seule table avec des problèmes de redondance.

Quel schéma conceptuel choisir (exemple) ?

Second schéma

```
-- Perte d'information
-- sauf si on introduit une contrainte ((ai,bi) = (a
SELECT A, B, C
FROM   (SELECT A, B FROM R) AS R1
  NATURAL JOIN
       (SELECT B, C FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;
```

Les tables $\pi[AB](R)$ et $\pi[BC](R)$: Pas de redondance mais une perte d'information.

Quel schéma conceptuel choisir (exemple) ?

Troisième schéma

```
-- La contrainte devient une clef primaire
-- La clef primaire devient une contrainte inter-re
SELECT A, B, C
FROM   (SELECT A, C FROM R) AS R1
  NATURAL JOIN
       (SELECT B, C FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;
```

Les tables $\pi[AC](R)$ et $\pi[BC](R)$: Pas de redondance mais une perte d'information.

Quel schéma conceptuel choisir (exemple) ?

Quatrième schéma

```
-- La contrainte ((ai = aj) => (ci = cj))
-- devient une clef primaire
SELECT A, B, C
FROM   (SELECT A, C FROM R) AS R1
  NATURAL JOIN
       (SELECT A, B FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;
```

Les tables $\pi[AB](R)$ et $\pi[AC](R)$: Ni redondance ni perte d'information.

Quel schéma conceptuel choisir (exemple) ?

Bilan

Les problèmes majeurs :

Redondance La relation R possède de l'information redondante sur les attributs A et C.

Perte d'information Les décompositions peuvent créer des n-uplets lors des jointures.

Perte de contraintes Les décompositions peuvent nécessiter la création de contraintes inter-relations.

Schéma conceptuel relationnel

Définition (rappel)

Un schéma relationnel se compose de :

- un ensemble de schémas de relations $\{Schema(R^i) =< (A^i_1, A^i_2, \ldots, A^i_{n_i}), \{I^i\}, P^i >\}$,
- pour chaque relation R^i, un ensemble (éventuellement vide) d'identifiants externes,
- une contrainte d'intégrité inter-relations (éventuellement *True*).

et les opérations classiques d'insertion, de suppression et de modification dans une base de données.

Schéma conceptuel relationnel

Les problèmes potentiels

Ils sont liés au besoin de vérifier les contraintes d'intégrité après :

insertion d'un n-uplet : présence dans la relation; unicité de son identifiant; éventuellement présence des identifiants externes et respect des contraintes d'intégrité, et enfin le respect de l'éventuelle contrainte d'intégrité inter-relations.

suppression d'un n-uplet : suppression des n-uplets *dépendants*. Ces dépendances sont les contraintes d'intégrités de la relation, les identifiants externes des autres relations et la contrainte inter-relations.

modification d'un n-uplet : modifications des n-uplets *dépendants*. Ces dépendances sont les contraintes d'intégrités de la relation, les identifiants externes des autres relations et la contrainte inter-relations.

Schéma conceptuel relationnel

Un “bon” schéma

Un bon schéma conceptuel est donc un schéma qui *minimise* le temps nécessaire à ces vérifications.
En pratique, il faut chercher à diminuer

1. les contraintes inter-relations qui sont les plus coûteuses,
2. les clefs étrangères qui génèrent les modifications et/ou les suppressions.

tout en conservant les données et les informations sémantiques.

Dépendances fonctionnelles et décomposition

Notations

Pour simplifier, on notera X l'ensemble d'attributs $\{X_1, \ldots, X_n\}$.
Sauf mention spéciale, on aura toujours $X \cap Y = \emptyset$.

Dépendances fonctionnelles et décomposition

Dépendance fonctionnelle (DF)

Definition

Soit $R(X, Y, Z)$ une relation.
$X \longrightarrow Y$ est une dépendance fonctionnelle (DF) si et seulement si $(x^1 = x^2) \Rightarrow (y^1 = y^2)$.

Intuitivement, si $X \longrightarrow Y$ est une DF, alors X est un déterminant (un identifiant) pour Y.

Dépendances fonctionnelles et décomposition

Décomposition sans perte d'information

Definition

Soit $R(X, Y, Z)$ une relation.
La décomposition de R en $R1(X, Y)$ et $R2(X, Z)$ est sans perte d'information si et seulement si $\pi[X, Y](R) \bowtie \pi[X, Z](R) = R$.

Dépendances fonctionnelles et décomposition

Théorème de Heath

Propriété

Soit $R(X, Y, Z)$ une relation.
La décomposition de R en $R1(X, Y)$ et $R2(X, Z)$ est sans perte d'information si il existe une DF $X \longrightarrow Y$.

Cela explique la bonne décomposition pour notre exemple.

Dépendances fonctionnelles et décomposition

Dépendance fonctionnelle irréductible (DFI)

Definition

Soit $R(X, Y, Z)$ une relation, et $X \longrightarrow Y$ une DF.

- $X \longrightarrow Y$ est une DFI à droite si et seulement si $Y = \{Y_i\}$,
- $X \longrightarrow Y$ est une DFI à gauche si et seulement $\forall X_i \ (X - \{X_i\}) \not\longrightarrow Y$.
- $X \longrightarrow Y$ est une DFI si et seulement si elle c'est une DFI à droite et une DFI à gauche.

Dépendances fonctionnelles et décomposition

Dépendance fonctionnelle déduite

Definition

la réflexivité : si $Y \subset X$ alors $X \longrightarrow Y$ est une DF déduite.

l'augmentation : Soit $X \longrightarrow Y$ une DF, alors
$\forall Z, (X \cup Z) \longrightarrow (Y \cup Z)$ est une DF déduite.

la transitivité : Soient $X \longrightarrow Y$ et $Y \longrightarrow Z$ deux DFs, alors
$X \longrightarrow Z$ est une DF déduite.

Graphes des dépendances fonctionnelles d'une relation

Représentation graphique de DFs

Soient une relation R sur les attributs $A = \{A_1, \ldots, A_n\}$, et un ensemble de DFs, $D_R = \{X \longrightarrow Y\}$.
Cet ensemble D_R défini un graphe $D(R)$ sur l'ensemble des parties de A.
Exemple : $R(A, B, C)$. Graphe $D(R)$ de
$D_R = \{\{A, B\} \longrightarrow \{C\}, \{A\} \longrightarrow \{C\}\}$.

Graphes des dépendances fonctionnelles d'une relation

Graphe des DFs d'une relation

Definition

Soit $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs avec $X \subseteq A$ et $Y \subseteq A$.
Le graphe $D^*(R)$ des dépendances fonctionnelles de R est le graphe obtenu par fermeture (clôture) du graphe $D(R)$ en ajoutant toutes les DFs déduites par réflexivité, augmentation et transitivité.

Exemple : $R(A, B, C)$. Graphe $D^*(R)$ de $D_R = \{\{A, B\} \longrightarrow \{C\}, \{A\} \longrightarrow \{C\}\}$.

Graphes des dépendances fonctionnelles d'une relation

Ensemble irréductible de DFs

Definition

Soit $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs avec $X \subseteq A$ et $Y \subseteq A$.
D_R est irréductible ssi toutes les DFs sont irréductibles et $\forall D'_R \subset D_R,\ D'^*(R) \neq D^*(R)$.

Graphes des dépendances fonctionnelles d'une relation

Graphe miminal des DFs

Propriété

Soit $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs avec $X \subseteq A$ et $Y \subseteq A$.
Il existe F_R irréductible tel que $F^*(R) = D^*(R)$.

Remarque : L'ensemble F_R n'est pas obligatoirement unique. Cette non unicité est le principal obstacle à une automatisation totale du processus.

Graphes des dépendances fonctionnelles d'une relation

Algorithme $D_R \rightarrow F_R$

1. Remplacer chaque DF non irréductible à droite par un ensemble équivalent de DFs irréductibles à droite (singleton) obtenus par réflexivité + transitivité.
2. Pour chaque DF, tester si la suppression d'un attribut A_i à gauche modifie la clôture. Si non, réduire la DF. Toutes les DFs obtenues sont irréductibles.
3. Pour chaque DF, tester si sa suppression modifie la clôture. Si non, enlever la DF. L'ensemble obtenu est irréductible.

Remarque : La non unicité du graphe minimal (de F_R) dépend de l'ordre des choix faits dans l'algorithme précédent. C'est le principal obstacle à une automatisation totale du processus.

Exemple fil rouge

La relation et les DFs

$FIRST(Fournisseur, Nom, Statut, Ville, Piece, Quantite)$ et ses dépendances fonctionnelles :

- $\{Fournisseur\} \longrightarrow \{Statut, Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece, Ville\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

Exemple fil rouge

L'algorithme $D \rightarrow F$ (1)

Remplacement des $\rightarrow$ non irréductibles à droite.

- $\{Fournisseur\} \longrightarrow \{Statut, Ville\}$
  - $\{Fournisseur\} \longrightarrow \{Statut\}$
  - $\{Fournisseur\} \longrightarrow \{Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece, Ville\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

Exemple fil rouge

L'algorithme $D \rightarrow F$ (2)

Suppression des attributs inutiles à gauche.

- $\{Fournisseur\} \longrightarrow \{Statut\}$
- $\{Fournisseur\} \longrightarrow \{Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece, Ville\} \longrightarrow \{Quantite\}$
  - $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

Exemple fil rouge

L'algorithme $D \rightarrow F$ (3)

Suppression des $\rightarrow$ déduites.

- $\{Fournisseur\} \longrightarrow \{Statut\}$
  -
- $\{Fournisseur\} \longrightarrow \{Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

Exemple fil rouge

L'algorithme $D \rightarrow F$ (resultat)

- $\{Fournisseur\} \longrightarrow \{Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

Formes normales

Clef candidate

Definition

Soient $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs avec $X \subseteq A$ et $Y \subseteq A$.

$C \subseteq A$ est une clef candidate de R ssi $(C \longrightarrow A) \in D_R^*$ et est irréductible à gauche.

Exemple : $\{Fournisseur, Piece\}$ et $\{Nom, Piece\}$ sont les clefs candidates de $FIRST$.

Formes normales

Première forme normale (1NF)

Problème 1 : Dans une table, quel est la signification de deux n-uplets égaux ?

Definition

Une *table* est en 1NF si et seulement si c'est une relation.

Remarque : Attention, SQL qui manipule en réalité des multi-ensembles peut donc avoir des tables qui ne sont pas en 1NF.

Propriété

FIRST est en 1NF ssi il n'y a pas de doublons.

Formes normales

Seconde forme normale (2NF)

Problème 2 : Dans *FIRST*, pour chaque pièce d'un fournisseur, la ville est renseignée. Cela engendre de la redondance et des difficultés de mise à jour.

Definition

Soient $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R. $R(A)$ est en 2NF si et seulement si :

- elle est en 1NF,
- si l'on pose $Z = A - \cup iC^i$, alors $\forall i,\ \forall Z_j \in Z\ C^i \longrightarrow \{Z_j\}$ est irréductible.

Informellement, les clefs candidates ne contiennent pas strictement de déterminants.

Formes normales

2NF : Exemple

Propriété

- *FIRST n'est pas en 2NF. En effet* $\{Fournisseur, Piece\}$ *est une clef candidate et* $\{Fournisseur, Piece\} \longrightarrow \{Ville\}$ *n'est pas irréductible.*
- $SECOND(Fournisseur, Statut, Ville)$ *et* $FNPQ(Fournisseur, Nom, Piece, Quantite)$ *sont en 2NF.*
- *Le théorème de Health appliqué à* $\{Fournisseur\} \longrightarrow \{Statut, Ville\}$ *indique que la décomposition de FIRST en SECOND et FNPQ est sans perte d'information.*

Formes normales

Troisième forme normale (3NF)

Problème 3 : Dans *SECOND*, pour chaque fournisseur d'une ville, le statut est identique. Cela engendre de la redondance et des difficultés de mise à jour.

Definition

Soient $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R. $R(A)$ est en 3NF si et seulement si :

- elle est en 2NF,
- si l'on pose $Z = A - \cup iC^i$, alors $\forall Z_i \in Z,\ (Z - \{Z_i\}) \not\longrightarrow \{Z_i\}$.

Informellement, les attributs qui n'appartiennent à aucune clef candidate sont indépendants. Informellement, il n'y a pas de dépendances transitives.

Formes normales

3NF : Exemple

Propriété

- $SECOND(Fournisseur, Statut, Ville)$ *n'est pas en 3NF. En effet* $\{Fournisseur\}$ *est l'unique clef candidate et* $\{Ville\} \longrightarrow \{Statut\}$
- $FV(Fournisseur, Ville)$, $VS(Ville, Statut)$ *et* $FNPQ(Fournisseur, Nom, Piece, Quantite)$ *sont en 3NF.*
- *Le théorème de Health appliqué à* $\{Ville\} \longrightarrow \{Statut\}$ *indique que la décomposition de SECOND est sans perte d'information.*

Formes normales

Forme normale de Boyce-Codd (BCNF)

Problème 4 : Dans *FNPQ*, pour chaque pièce, le fournisseur détermine le nom et réciproquement. Cela engendre de la redondance et des difficultés de mise à jour.

Definition

Soient $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R. $R(A)$ est en BCNF si et seulement si :

- elle est en 1NF,
- si $X \longrightarrow Y$ est irréductible à gauche et $X \cap Y = \emptyset$, alors $\exists i, C^i = X$

Informellement, les déterminants de telles DFs sont des clefs candidates.

Formes normales

BCNF : Exemple

Propriété

- *FNPQ*(*Fournisseur*, *Nom*, *Piece*, *Quantite*) *n'est pas en BCNF. En effet* $\{Nom\}$ *n'est pas une clef candidate et* $\{Nom\} \longrightarrow \{Fournisseur\}$
- *FN*(*Fournisseur*, *Nom*) *et FPQ*(*Fournisseur*, *Piece*, *Quantite*) *sont en BCNF.*
- *Le théorème de Health appliqué à* $\{Fournisseur\} \longrightarrow \{Nom\}$ *indique que la décomposition de FNPQ est sans perte d'information.*

Formes normales

Exemple : Bilan

- $FIRST = FV \bowtie VS \bowtie FNPQ$ toutes 3NF.
- $FIRST = FV \bowtie VS \bowtie FN \bowtie FPQ$ toutes BCNF.
- Les décompositions obtenues ne sont pas idéales. On peut regrouper FV et FN en une seule relation FNV également BCNF.

Formes normales

Perte de dépendances fonctionnelles (1)

Soient la relation $EMP(Eleve, Matiere, Professeur)$ et les dépendances fonctionnelles :

- $\{Eleve, Matiere\} \longrightarrow \{Professeur\}$
- $\{Professeur\} \longrightarrow \{Matiere\}$

qui traduisent le fait qu'un professeur n'enseigne qu'une matière, et qu'un élève n'a qu'un professeur pour une matière donnée.

Formes normales

Perte de dépendances fonctionnelles (1)

Propriété

- $C = \{\{E, M\}, \{E, P\}\}$,
- $A = \cup C_i$, *donc EMP est en 2NF et 3NF.*
- $P \longrightarrow M$ *est irréductible et* $\forall C_i, C_i \neq \{P\}$ *donc EMP n'est pas en BCNF.*
- $PM(Professeur, Matiere)$ $EP(Eleve, Professeur)$ *sont en BCNF, mais la dépendance fonctionnelle* $\{Eleve, Matiere\} \longrightarrow \{Professeur\}$ *est devenue une contrainte inter-relation.*
- *Aucune des deux décompositions (3NF ou BCNF) n'est satisfaisante. 3NF du fait de la redondance et BCNF du fait de la perte de DFs.*

Formes normales

Normalisation : Bilan

- BCNF $\Rightarrow$ 3NF. (démonstration possible en TD)
- Les formes normales forment une hiérarchie.
  $BCNF \subset 3NF \subset 2NF \subset 1NF$
- Cette hiérarchie continue $5NF \subset 4NF \subset BCNF$
- Théorie basée sur la projection-jointure. Il en existe d'autres (une est basée sur la sélection-union).
- Toute relation 1NF est décomposable en un ensemble de relations 3NF sans perte d'informations et sans perte de dépendances fonctionnelles.
- Toute relation 1NF est décomposable en un ensemble de relations BCNF sans perte d'informations, mais éventuellement avec perte de dépendances fonctionnelles.
- Les décompositions obtenues ne sont pas toujours "optimales".

L'utilisateur doit faire des choix.

Algorithmes de normalisation

Données en entrée

Soient $R(A)$ une relation et $D_R = \{X \longrightarrow Y\}$ un ensemble de DFs.

Algorithmes de normalisation

Algo de décomposition en 3NF

1. Préliminaires
2. Décomposition
3. Post-Traitement

Algorithmes de normalisation

Algo de décomposition en 3NF (Préliminaires)

1. Calculer $I = irreductible(DF)$. (I n'est pas unique)
2. Calculer $C^1, \ldots C^p$ les clefs candidates de R.

Algorithmes de normalisation

Algo de décomposition en 3NF (Décomposition)

1. $D = \emptyset$. (D contiendra l'ensemble des relations)
2. Répéter
   - 2.1 Choisir $f \in I, f = X \longrightarrow Y_f$. (influence de l'ordre)
   - 2.2 Calculer $F = \{g \in I, g = X \longrightarrow Y_g\}$.
   - 2.3 Calculer $Z = \cup_g Y_g$.
   - 2.4 Calculer $D = D \cup \{\pi[X, Z](R)\}$.
   - 2.5 Calculer $I = I - (F \cup \{fd \in (X \cup Z)\})$. (utile si $X \longrightarrow Y$ et $Y \longrightarrow X$)

   jusqu'à $I = \emptyset$.

Algorithmes de normalisation

Algo de décomposition en 3NF (Post-Traitement)

1. Calculer $B = A - \cup Att(D_i)$ (les attributs manquants).
2. Si $B \neq \emptyset$ alors $D = D \cup \pi[C^i, B](R)$.
3. Si aucune relation $d \in D$ ne contient de clefs candidate C^i, alors choisir une clef C^i et faire $D = D \cup \pi[C^i](R)$. (le choix modifie le résultat)
4. Si les attributs d'une relation sont inclus dans ceux d'une autre, supprimer la relation.

Algorithmes de normalisation

Algo de décomposition en 3NF (Exemple)

Les résultats possibles pour D :

1. $\{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]$
2. $\{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Fo$
3. $\{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
4. $\{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Nom, Vill$

Algorithmes de normalisation

Algo de décomposition en BCNF

1. Décomposition
2. Post-Traitement

Algorithmes de normalisation

Algo de décomposition en BCNF (Décomposition)

1. Initialiser $D = \{R\}$.
2. Pour chaque $T \in D$
   - 2.1 Calculer $I^T = irreductible(DF(T))$. (non unique)
   - 2.2 Calculer $C_T^1, \ldots C_T^p$ les clefs candidates de T.
   - 2.3 Trouver $f \in I^T, f = X \longrightarrow Y_f$ et $X \neq C_T^i$. (T non BCNF)
   - 2.4 Si trouve alors
     - 2.4.1 Calculer $F = \{g \in I^T, g = X \longrightarrow Y_g\}$.
     - 2.4.2 Calculer $Z = \cup_g Y_g$.
     - 2.4.3 Calculer $D = (D - \{T\}) \cup \{\pi[X, Z](T), \pi[A^T - Z](T)\}$.

Algorithmes de normalisation

Algo de décomposition en BCNF (Post-Traitement)

1. Calculer $I^R = irreductible(DF(R))$. ($I^R$ n'est pas unique)
2. Pour chaque $f \in I^R, f = X \longrightarrow Y_f$ faire
   - 2.1 si $(X \cup Y_f) \subseteq Att(R_i)$ alors associer f à R_i
   - 2.2 sinon créer une contrainte inter-relations.

Algorithmes de normalisation

Algo de décomposition en BCNF (Exemple)

Les résultats possibles pour D :

1. $\{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]$
2. $\{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Fo$
3. $\{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
4. $\{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Nom, Vill$

Algorithmes de normalisation

Algos de décomposition (des outils ?)

Problème : Il est possible de construire un outil qui propose toutes les solutions. Mais sur une *vraie* BD, le nombre peut être très grand, et le temps de calcul assez long. À l'inverse, un outil qui renvoie une seule proposition risque de retourner une solution non satisfaisante pour le concepteur.

Intérêt de la normalisation

Objectifs

La théorie de la normalisation formalise des règles de bon sens que tout bon concepteur applique plus ou moins inconsciemment.
L'intérêt est double :

- légitimer le bon sens,
- permettre le développement d'outils.

En pratique

Il faut chercher à obtenir une décomposition en BCNF. Si il y a perte de dépendances fonctionnelles, c'est au concepteur de choisir entre :

- BCNF avec des contraintes inter-relations,
- décomposition en 3NF qui possède éventuellement quelques redondances et donc des problèmes de maintenance.

Modèle conceptuel Entité-Association et normalisation

Le modèle Entité-Association (définition)

La plupart des méthodes (Merise, ...) utilisées en conception de bases de données préfère le modèle conceptuel Entité-Association au modèle conceptuel relationnel.
Schématiquement le modèle Entité-Association comprend :

Des entités qui correspondent aux *objets* de la base. Les caractéristiques des objets sont les attributs.

Des associations qui lient les objets.

Modèle conceptuel Entité-Association et normalisation

Le modèle Entité-Association (méthodologie)

- Pour chaque objet, créer une relation. Si la clef primaire est composé, ajouter un identifiant unique de l'objet qui devient alors la clef primaire.
- Pour chaque association, créer une relation qui contient uniquement les clefs primaires de tous les objets en relation.

Le modèle Entité-Association (résultat)

Ces méthodologies très simples sont efficaces dans la grande majorité des situations car elles conduisent naturellement à des relations 3NF et même souvent BCNF.

Outline

Les calculs relationnels

Algèbre relationnelle

Avec l'algèbre relationnelle, une requête est exprimée comme une séquence d'opérations à réaliser à partir des relations de la bd.

Langages prédicatifs

Les langages prédicatifs (ou langage de type calcul) exprime une requête par la simple définition du résultat.
Ce sont des langages déclaratifs.

Le calcul de tuples (définition)

La partie déclarative

Elle associe les variables aux relations :

- $x \in R$, x désignera les tuples de R.
- $x \in (S \cup T)$, x désignera les tuples de S ou bien de T.

Le calcul de tuples (définition)

Les formules booléennes valides

Elles sont contruites à partir des règles :

R1 *true* et *false* sont des formules valides.

R2 une condition : $x.A\,\theta\,a$ ou bien : $x.A\,\theta\,y.B$ avec a une constante et θ un opérateur parmi $\{=, \neq, <, \leq, >, \geq\}$, est une formule valide.

R3 si f, f_1, f_2 sont des formules valides, alors $\{(f), \neg f, f_1 \vee f_2, f_1 \wedge f_2\}$ sont des formules valides.

R4 si f_x est une formule valide où x est une variable libre, alors $\exists x(f_x)$ et $\forall x(f_x)$ sont des formules valides où x est une variable liée.

R5 Il n'y a pas d'autre formule valide.

Le calcul de tuples (définition)

Les expressions

Elles sont de la forme $\{x.A_i^x, y.A_j^y, \ldots, z.A_k^z | f_{(x,y,\ldots,z,t,u,\ldots,w)}\}$ où :

- $\{x, y, \ldots, z\}$ sont des variables déclarées de $R^x, R^y, \ldots, R^z$.
- $A_i^x, A_j^y, \ldots, A_k^z$ sont des attributs des relations correspondantes.
- $f_{(x,y,\ldots,z,t,u,\ldots,w)}$ est une formule valide où les variables $\{t, u, \ldots, w\}$ sont liées par des quantificateurs.

La sémantique d'une expression

C'est la projection sur les attributs $A_i^x, A_j^y, \ldots, A_k^z$ du produit cartésien des relations $R^x, R^y, \ldots, R^z$ réduit aux tuples pour lesquels la formule $f_{(x,y,\ldots,z,t,u,\ldots,w)}$ est vraie.

Le calcul de tuples (définition)

Propriété

On peut montrer que le calcul de tuples à la même pouvoir expressif que l'algèbre relationnelle.

Remarque

Le langage QUEL (défini en 1975 pour le système INGRES) est basé sur le calcul de tuples.

Le calcul de domaines (définition)

Similaire au calcul de tuples, mais les variables portent sur les valeurs des attributs et non sur les tuples.

La partie déclarative

Elle associe les variables aux **attributs** :

- $x \in R.A$, x désignera les valeurs de A.
- $x \in (S.A \cup T.B)$, x désignera les valeurs de $S.A$ ou bien de $T.B$.

Le calcul de domaines (définition)

Les formules booléennes valides

Elles sont contruites à partir des règles :

R1 *true* et *false* sont des formules valides.

R2 une condition : $x\ \theta\ a$ ou bien : $x\ \theta\ y$ avec a une constante et θ un opérateur parmi $\{=, \neq, <, \leq, >, \geq\}$, est une formule valide.

R3 une condition d'appartenance $Exist_R(A_1 : v_1, A_2 : v_2, \ldots, A_k : v_k)$ est une formule valide. Elle est vraie si et seulement si il existe au moins un tuple ayant la bonne valeur dans R.

R4 si f, f_1, f_2 sont des formules valides, alors $\{(f), \neg f, f_1 \vee f_2, f_1 \wedge f_2\}$ sont des formules valides.

R5 si f_x est une formule valide où x est une variable libre, alors $\exists x(f_x)$ et $\forall x(f_x)$ sont des formules valides où x est une variable liée.

R6 Il n'y a pas d'autre formule valide.

Le calcul de domaines (définition)

Les expressions

Elles sont de la forme $\{x, y, \ldots, z | f_{(x,y,\ldots,z,t,u,\ldots,w)}\}$ où :

- $\{x, y, \ldots, z\}$ sont des variables déclarées de $R_x.A_x, R_y.A_y, \ldots, R_z.A_z$.
- $f_{(x,y,\ldots,z,t,u,\ldots,w)}$ est une formule valide dans laquelle les variables $\{t, u, \ldots, w\}$ sont liées par des quantificateurs.

La sémantique d'une expression

C'est l'ensemble des tuples $< x, y, \ldots, z >$ du produit cartésien des relations $R_x, R_y, \ldots, R_z$ réduit aux tuples pour lesquels la formule $f_{(x,y,\ldots,z,t,u,\ldots,w)}$ est vraie.

Le calcul de domaines (définition)

Propriété

On peut montrer que le calcul des domaines à la même pouvoir expressif que l'algèbre relationnelle.

Remarque

Le langage QBE (Query By Example défini par IBM) est basé sur le calcul des domaines.

Les calculs de tuples et domaines (exemples)

La sélection

$$\sigma[(\texttt{Nom} = \texttt{Nom2})|(\texttt{Datedenaissance} = \texttt{31mars1985})](\texttt{Etudiants})$$

s'écrit en calcul de tuples

$$\{e : \mathit{Etudiants}, e.\mathit{Numero}, e.\mathit{Nom}, e.\mathit{Prenom}, e.\mathit{DateDeNaissance}$$
$$\texttt{tel que } e.\mathit{Nom} = \texttt{Nom2} \vee e.\mathit{DateDeNaissance} = \texttt{31mars1985}\}$$

et en calcul de domaines

$$\{\mathit{num} : \mathit{Etudiants}.\mathit{Numero}, \mathit{nom} : \mathit{Etudiants}.\mathit{Nom},$$
$$\mathit{num} : \mathit{prenom} : \mathit{Etudiants}.\mathit{Prenom}, \mathit{date} : \mathit{Etudiants}.\mathit{DateDeNaissance}$$
$$\texttt{tel que } \mathit{Exist}_{\mathit{Etudiants}}(\mathit{num}, \mathit{nom}, \mathit{prenom}, \mathit{date})$$
$$\wedge(\mathit{nom} = \texttt{Nom2} \vee \mathit{date} = \texttt{31mars1985})\}$$

Les calculs de tuples et domaines (exemples)

La projection

$$\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$$

s'écrit en calcul de tuples

$$e : \mathit{Etudiants},\ \{e.\mathit{Nom}, e.\mathit{Prenom}\ \texttt{tel que}\ \mathit{true}\}$$

et en calcul de domaines

$$\{\mathit{nom} : \mathit{Etudiants}.\mathit{Nom},\ \mathit{prenom} : \mathit{Etudiants}.\mathit{Prenom}$$
$$\texttt{tel que}\ \exists \mathit{num} : \mathit{Etudiants}.\mathit{Numero}, \exists \mathit{date} : \mathit{Etudiants}.\mathit{DateDeNaissance},$$

Les calculs de tuples et domaines (exemples)

Les étudiants ayant au moins deux prénoms

s'écrit en calcul de tuples

$$e : Etudiants\{e.Nom$$

$$\texttt{tel que } \exists x : AutresPrenoms(e.Numero = x.Numero)\}$$

et en calcul de domaines

$\{nom : Etudiants.Nom \texttt{ tel que}$
$\exists num : Etudiants.Numero,\ \exists prenom : Etudiants.Prenom,$
$\exists date : Etudiants.DateDeNaissance,\ \exists num2 : AutresPrenoms.Numero,$
$\exists prenom2 : AutresPrenoms.Prenom,\ \exists ordre : AutresPrenoms.Ordre,$
$Exist_{Etudiants}(num, nom, prenom, date)$
$\wedge Exist_{AutresPrenoms}(num2, prenom2, ordre)$
$\wedge (num = num2)\}$

Outline

SQL : un langage prédicatif

SQL : Structured Query Language

- SQL permet d'exprimer tous les opérateurs de l'algèbre relationnelle.
- SQL permet également l'écriture de requêtes sous forme déclarative. SQL permet le calcul de domaine, et la déclaration des variables est implicite.

Ainsi l'utilisateur peut choisir.

SQL : un langage prédicatif

Forme générale d'une requête SQL

```
SELECT liste1Attributs
FROM   R1, R2, ..., Rn
WHERE  conditionDeJointure
  AND  expression;
```

Les conditions sur les relations

L'appartenance d'un attribut à un ensemble

- Le mot clef `IN` correspond à l'opérateur mathématique $\in$ de la théorie des ensembles.
- **Restriction :** L'ensemble doit contenir des valeurs *atomiques*.

Les conditions sur les relations

L'appartenance d'un attribut à un ensemble (Exemple 1)

```
-- Calcul des étudiants ayant au moins deux prenoms
SELECT *
FROM   Etudiants
WHERE  Numero IN
       (SELECT Numero
        FROM   AutresPrenoms);
```

Les conditions sur les relations

L'appartenance d'un attribut à un ensemble (Exemple 2)

```
-- Les étudiants ayant au moins deux prénoms
-- ou un prénom qui soit un autre prénom pour un étu
SELECT *
FROM   Etudiants
WHERE  Numero IN
       (SELECT Numero
        FROM   AutresPrenoms)
   OR  Prenom IN
       (SELECT Prenom
        FROM   AutresPrenoms);
```

Les conditions sur les relations

La comparaison d'un attribut à un ensemble

- Les mots clefs `ANY` et `SOME` correspondent à l'existence dans l'ensemble d'au moins un élément satisfaisant la condition.
- Le mot clef `ALL` correspond au fait que tous les éléments de l'ensemble satisfont la condition.
- **Restriction :** Les ensembles doivent contenir des valeurs *atomiques*.

Les conditions sur les relations

La comparaison d'un attribut à un ensemble (Exemple 1)

```
-- Calcul des étudiants ayant au moins deux prénoms
SELECT *
FROM   Etudiants
WHERE  Numero = ANY -- idem = SOME et idem IN dans
       (SELECT Numero
        FROM   AutresPrenoms);
```

Les conditions sur les relations

La comparaison d'un attribut à un ensemble (Exemple 2)

```
-- Les étudiants dont le prénom est également un sec
SELECT *
FROM   Etudiants
WHERE  Prenom = SOME
       (SELECT Prenom
        FROM   AutresPrenoms
        WHERE  Ordre = 2);
```

Les conditions sur les relations

La comparaison d'un attribut à un ensemble (Exemple 3)

```
-- Les étudiants dont le prénom n'est le second prén
SELECT *
FROM   Etudiants
WHERE  Prenom <> ALL -- idem NOT Prenom = ANY
       (SELECT Prenom
        FROM   AutresPrenoms
        WHERE  Ordre = 2);
```

Les conditions sur les relations

Les comparaisons entre ensembles

- Le mot clef `CONTAINS` correspond à l'opérateur mathématique $\supseteq$ de la théorie des ensembles.
- L'égalité $=$ est également une comparaison possible.
- **Restriction :** Les ensembles doivent contenir **un seul** tuple et le premier doit être déclaré par le mot clef **ROW** comme un tuple.

Les conditions sur les relations

Les comparaisons entre ensembles (Exemple 1)

```
-- Les homonymes
SELECT *
FROM   Etudiants
WHERE  ROW(Nom, Prenom)
       =
          (SELECT DISTINCT Nom, Prenom
           FROM   Etudiants AS TEMP
           WHERE  Etudiants.Numero <> TEMP.Numero
             AND  Etudiants.Nom = TEMP.Nom
             AND  Etudiants.Prenom = TEMP.Prenom);
```

Les conditions sur les relations

Les comparaisons entre ensembles (Exemple 2)

```
-- Calcul des étudiants dont les prénoms sont inclus
/*
 * CONTAINS n'est pas implémenté dans PostGres
SELECT *
FROM   Etudiants
WHERE  ((SELECT Prenom
         FROM   AutresPrenoms
         WHERE  Ordre = 2)
       CONTAINS
-- ou = ou <>
        (SELECT Prenom
         FROM   AutresPrenoms
         WHERE  Numero = Etudiants.Numero));
*/
```

Les conditions sur les relations

L'existence d'un tuple dans une relation

Le mot clef `EXISTS` retourne `vrai` si et seulement si la relation est non vide.

Les conditions sur les relations

L'existence d'un tuple dans une relation (Exemple 1)

```
-- Les étudiants ayant au moins deux prénoms
SELECT *
FROM   Etudiants
WHERE  EXISTS
       (SELECT *
        FROM   AutresPrenoms
        WHERE  Etudiants.Numero = AutresPrenoms.Num
```

Les conditions sur les relations

L'existence d'un tuple dans une relation (Exemple 2)

```
-- Les étudiants ayant un seul prénom
SELECT *
FROM   Etudiants
WHERE  NOT EXISTS
       (SELECT *
        FROM   AutresPrenoms
        WHERE  Etudiants.Numero = AutresPrenoms.Num
```

Et la division ?

L'exemple (R1)

Celui traité lors de la division SQL algébrique.

```
CREATE TABLE R1 (
   lettre   char(1),
   chiffre  integer,
   indice   integer,
   pays     char(1),
PRIMARY KEY (lettre, chiffre, indice, pays)
);

INSERT INTO R1 VALUES ('A', 7, 33, 'F');
INSERT INTO R1 VALUES ('A', 7, 39, 'I');
INSERT INTO R1 VALUES ('A', 7, 34, 'G');
INSERT INTO R1 VALUES ('F', 6, 39, 'I');
INSERT INTO R1 VALUES ('C', 4, 33, 'F');
INSERT INTO R1 VALUES ('C', 4, 39, 'I');
```

Et la division ?

L'exemple (R2)

```
CREATE TABLE R2 (
   indice   integer,
   pays     char(1),
PRIMARY KEY (indice, pays)
);

INSERT INTO R2 VALUES (33, 'F');
INSERT INTO R2 VALUES (39, 'I');
```

Et la division ?

Division avec `CONTAINS`

Soient $R1(X, Y)$ et $R2(Y)$. $R1 \div R2$ est égal à :

$$R3 = \{x \in R1.X \ / \ \pi[Y](\sigma[X = x](R1)) \supseteq R2\}$$

.

Et la division ?

Division avec CONTAINS (Exemple)

```
-- La division basee sur R3={x in R1.X / Proj[Y](Se
-- les couples (lettre, chiffre) pour lesquels l'as
-- tous les tuples de R2 forment des tuples de R1
/*
SELECT DISTINCT lettre, chiffre
FROM   R1
WHERE  ((SELECT indice, pays
         FROM   R1 AS R11
         WHERE  R11.lettre  = R1.lettre
           AND  R11.chiffre = R1.chiffre)
        CONTAINS
        (SELECT *
         FROM   R2));
*/
```

Malheureusement CONTAINS est rarement implémenté.

Et la division ?

Division avec $\forall$

$$R3 = \{x \in R1.X \ / \ \forall y \in R2, \ \exists (a, b) \in R1 \ \wedge \ (x = a \wedge y = b)\}$$

Malheureusement le quantificateur $\forall$ n'est jamais implémenté.

Et la division ?

Division avec $\exists$

Nous savons que $\forall x,\ f(x) \equiv \nexists x,\ \neg f(x)$, donc :

$$R3 = \{x \in R1.X \ /\ \ \nexists y \in R2,\ \nexists (a,b) \in R1\ \wedge\ (x = a \wedge y = b)\}$$

Et la division ?

Division avec ∃ (Exemple)

```
-- La division basee sur R3={x / NotExists y in R2, NotE
-- les couples (lettre, chiffre) pour lesquels l'associa
-- tous les tuples de R2 forment des tuples de R1
SELECT DISTINCT lettre, chiffre
FROM   R1
WHERE  NOT EXISTS
       (SELECT *
        FROM   R2
        WHERE  NOT EXISTS
               (SELECT indice, pays
                FROM   R1 AS R11
                WHERE  R11.lettre  = R1.lettre
                AND    R11.chiffre = R1.chiffre
                AND    R11.indice  = R2.indice
                AND    R11.pays    = R2.pays));
```

Outline

SQL : un langage de présentation des données

SQL : Structured Query Language

- SQL permet (un peu) de spécifier dans les requetes une mise en forme du résultat.
- Cela répond (un peu) à l'étape 8 du schéma global vu dans le chapitre introductif.
- Dans ce chapitre, nous ne mentionnons que quelques fonctionnalités du langage SQL.

Les fonctions d'agrégation

Les fonctions usuelles

cardinal : le mot clef `COUNT` opère sur un ensemble de tuples.

moyenne : le mot clef `AVERAGE` opère sur un ensemble de valeurs prises par un attribut.

minimum : le mot clef `MIN` opère sur un ensemble de valeurs prises par un attribut.

maximum : le mot clef `MAX` opère sur un ensemble de valeurs prises par un attribut.

total : le mot clef `SUM` opère sur un ensemble de valeurs prises par un attribut.

PostGres offre de nombreuses autres fonctions d'agrégation (cf doc).

Partitionnement des tuples d'une relation

Le mot clef `GROUP BY`

Il permet de partitionner l'ensemble des tuples d'une relation.

```
-- Le nombre d'étudiants pour chaque premier prénom
SELECT   Prenom, COUNT(DISTINCT Numero)
FROM     Etudiants
GROUP BY Prenom;
```

Partitionnement des tuples d'une relation

Le mot clef `HAVING`

Il permet de sélectionner dans une partition seulement certains groupes. **Attention** la condition porte sur le groupe et non sur les tuples.

```
-- Le nombre d'étudiants pour chaque premier prénom
SELECT   Prenom, COUNT(DISTINCT Numero)
FROM     Etudiants
GROUP BY Prenom
HAVING   MIN(DateDeNaissance) > 1980-1-1;
```

L'ordonnancement des tuples

Le mot clef `ORDER BY`

SQL manipule des relations, mais l'utilisateur souhaite souvent mettre en page le résultat d'une requête à des fins de lisibilité.
`ORDER BY` permet de trier les tuples du résultat d'une requête.

```
-- Les étudiants par ordres alphabétiques nom, préno
SELECT   *
FROM     Etudiants
ORDER BY Nom, Prenom, DateDeNaissance DESC, Numero;
```

Autres fonctionnalités

L'aide en ligne

SQL possède de très nombreuses autres fonctionnalités qui ne seront pas détaillées car elles sortent du cadre de ce cours, mais que vous pouvez (devez) apprendre grâce notamment à l'aide en ligne de PostGres.

Autres fonctionnalités

Exemple : manipulation des dates

```
-- Les étudiants nés dans la nuit du 3 au 13 septem
SELECT    *
FROM      Etudiants
WHERE     DateDeNaissance BETWEEN '1752-9-4'
                          AND '1752-9-12';
```

Outline

Gestion de la correction : les transactions

Cohérence et correction : Concepts

Definition (État cohérent)

Une base de données est cohérente (ou dans un état cohérent) lorsque toutes les données qu'elle contient sont en accord avec les contraintes d'intégrité du schéma conceptuel.

Definition (État correct)

Une base de données est dans un état correct lorsque qu'elle est dans un état cohérent et que les valeurs des données reflètent exactement le résultat attendu des modifications effectuées.

Gestion de la correction : les transactions

Cohérence et correction : Propriétés

Propriété

Dans un SGBD relationnel, toute requête conserve la cohérence d'une base de données. Si elle s'exécute avec succès, elle transforme l'état cohérent en un nouvel état cohérent; si elle échoue (erreur de syntaxe, violation contrainte, bug SGBD, arrêt machine . . .), elle laisse la base dans son état cohérent.

Propriété

Dans un SGBD relationnel, les requêtes (même en cas de succès) ne conservent pas la correction d'une base de données.

Gestion de la correction : les transactions

Cohérence et correction : Exemple (1)

```
CREATE TABLE Client (
   nom      char(30),
   compte1  integer,
   compte2   integer,
PRIMARY KEY (nom)
);

CREATE TABLE Compte (
   numero   integer,
   valeur   integer,
PRIMARY KEY (numero)
);
```

Gestion de la correction : les transactions

Cohérence et correction : Exemple (2)

```
INSERT INTO Client VALUES ('Dupont', 1, 2);

INSERT INTO Compte VALUES (1, 3000);
INSERT INTO Compte VALUES (2, 500);

-- Transfert du compte 1 vers le compte 2 de 1000 e
UPDATE Compte SET valeur = valeur + 1000
  WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000
  WHERE numero = 1;
```

Entre les deux mises à jour, la base de données est cohérente mais non correcte.

La transaction : une unité logique de travail

Unité logique de travail

Definition

Une unité logique de travail est une suite ordonnée de requêtes qui conserve la correction des données d'une base. C'est une notion sémantique qui ne peut être que définie par l'utilisateur, et qui ne peut pas être définie comme une contrainte d'intégrité.

La transaction : une unité logique de travail

Transaction : définition

Definition

Une transaction est une suite ordonnée de requêtes qui se termine de deux façons possibles :

Succès Toutes les requêtes de la liste se sont exécutées avec succès. Le SGDB exécute une instruction `COMMIT` qui *valide* la transaction.

Échec Au moins une requête de la liste ne s'est pas exécutée avec succès. Le SGDB exécute une instruction `ROLLBACK` qui *invalide* la transaction et remet la base dans l'état du début de la transaction.

La transaction : une unité logique de travail

Transaction : exemple

```
-- Transfert du compte 1
-- vers le compte 2 de 1000 euros
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte
   SET valeur = valeur + 1000
   WHERE numero = 2;
UPDATE Compte
   SET valeur = valeur - 1000
   WHERE numero = 1;
END TRANSACTION ; -- ou bien END simplement
```

La transaction : une unité logique de travail

Transaction : propriété

Propriété

Dans un SGBD (relationnel), les transactions conservent la correction d'une base de données.

La transaction : une unité logique de travail

Transaction : remarque

Soit la requête

```
UPDATE Compte SET valeur = valeur + 1000;
```

qui modifie tous les tuples de la relation.

Si un incident système survient *au milieu* de la requête, la base de donnée sera dans un état incorrect.

Une solution consiste à considérer toutes les requêtes comme des transactions. C'est ce que font par défaut les SGBD (ils sont en mode *autocommit*).

La gestion des transactions

Les problèmes

Avec la définition précédente des transactions, deux problèmes majeurs subsistent :

1. Comment mettre en œuvre le `ROLLBACK` ?
2. En cas de panne, comment restaurer la base de données dans un état correct ?

La gestion des transactions

Journal et points de validation

Une solution commune aux deux problèmes est basée sur la gestion d'un journal. Un journal est composé principalement de deux choses :

- une redondance non pas des données mais des opérations effectuées sur les données. Seule les requêtes qui modifient le contenu sont enregistrées et souvent c'est le couple (valeurs avant, valeurs après) qui est mémorisé.
- de points de validation (ou points de synchronisation) qui définissent les états corrects de la base.

Le journal est manipulé à l'aide de deux opérations :

- `REDO` qui permet *de simuler* l'effet d'une requête grâce à la redondance du journal.
- `UNDO` qui permet *d'annuler* l'effet d'une requête grâce à la redondance du journal.

La gestion des transactions

Mise en œuvre d'une transaction : exemple (1)

```
-- Transfert du compte 1 vers le compte 2 de 1000 e
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte SET valeur = valeur + 1000
  WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000
  WHERE numero = 1;
END TRANSACTION ; -- ou bien END simplement
```

```
-- Transfert du compte 1 vers le compte 2 de 1000 e
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte SET valeur = valeur + 1000
  WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000
  WHERE XXX = 1;    -- erreur d'attribut
END TRANSACTION ; -- ou bien END simplement
```

La gestion des transactions

Mise en œuvre d'une transaction : exemple (2)

Pendant l'exécution, écriture dans le journal.

```
START TRANSACTION : ecrire(journal, "START␣TRANSACTION")
UPDATE : ecrire(journal, "Compte,␣(valeur,␣2)␣->␣(valeur
UPDATE : ecrire(journal, "Compte,␣(valeur,␣1)␣->␣(valeur
END TRANSACTION : COMMIT : ecrire(journal, "Etat␣corre
START TRANSACTION : ecrire(journal, "START␣TRANSACTION")
UPDATE : ecrire(journal, "Compte,␣(valeur,␣2)␣->␣(valeur
UPDATE : ROLLBACK :
UNDO : ecrire(journal, "Compte,␣(valeur,␣2)␣->␣(valeur
END TRANSACTION : COMMIT : ecrire(journal, "Etat␣corre
```

La gestion des transactions

Mise en œuvre d'une transaction : remarque

En fait le processus est un peu plus complexe du fait que les opérations se font en mémoire vive et que les écritures sur disques ne sont faites que lors des points de synchronisations (ou bien lorsque les tampons sont pleins).
Cela implique que l'écriture physique du journal soit faite avant l'enregistrement physique des données, et cela simplifie le traitement du `ROLLBACK`.

La gestion des transactions

La reprise après une panne : le problème

Soit la situation suivante

```
START TRANSACTION : ecrire(journal, "START␣TRANSACTION")
UPDATE : ecrire(journal, "Compte,␣(valeur,␣2)␣->␣(valeur
UPDATE : ecrire(journal, "Compte,␣(valeur,␣1)␣->␣(valeur
END TRANSACTION : COMMIT : ecrire(journal, "Etat␣corre
START TRANSACTION : ecrire(journal, "START␣TRANSACTION")
UPDATE : ecrire(journal, "Com
```

L'information dans le journal indique que la première transaction s'est déroulée normalement, mais pas la seconde. Elle n'indique pas le lien entre l'état de la mémoire principale et les valeurs enregistrées sur disque.
S'il est clair qu'il ne faut pas *valider* la seconde transaction, faut-il ou bien ne faut-il pas refaire la première transaction à partir de la sauvegarde restaurée ?

La gestion des transactions

La reprise après une panne : une solution

Une solution est basée sur les points de contrôles.
Un point de contrôle est une marque dans le journal qui liste toutes les transactions en cours, et qui garanti qu'à cet instant, les valeurs en mémoire et sur disque sont les mêmes.
Suivant les SGBD, les points de contrôle sont faits à date fixe, ou bien après un nombre déterminés de points de validation.

La gestion des transactions

La reprise après une panne : l'algorithme

L'algorithme de reprise après une panne est :

1. Restaurer la base.
2. À partir du dernier point de contrôle *pc*
   - 2.1 Initialiser deux ensembles $Undo = \texttt{transactions(pc)}$ et $Redo = \emptyset$
   - 2.2 À chaque `START TRANSACTION` T_i, faire $Undo = Undo \cup \{T_i\}$
   - 2.3 À chaque `COMMIT` T_i, faire $Redo = Redo \cup \{T_i\}$ et $Undo = Undo - \{T_i\}$
3. Parcourir en arrière le journal jusqu'à *pc* en effectuant les `UNDO` nécessaires.
4. Parcourir en avant le journal en effectuant les `REDO` nécessaires.

La gestion des transactions

La reprise après une panne : exemple

Soit un point de contrôle pc à la date tc et une défaillance du système à une date $tf > tc$. Il existe cinq types de transactions :

- T1 Les transactions commencées et finies avant tc.
- T2 Les transactions commencées avant tc et finies avant tf.
- T3 Les transactions commencées avant tc et non terminées.
- T4 Les transactions commencées et finies avant tf.
- T5 Les transactions commencées avant tf et non terminées.

L'algorithme retourne $Undo = \{T3, T5\}$ et $Redo = \{T2, T4\}$.

Outline

La gestion des accès concurrents

SGBD : accès simultanés à la BD

Les SGBD ont étés créés avec comme objectifs de permettre à des applications multiples d'accéder de façons uniforme à des données qui peuvent être distribuées. Une des conséquences est l'accès concurrent aux données.

Les problèmes potentiels

Les dépendances non valides dues à un échec (1)

Transaction A	temps	Transaction B
	t1	update $<$ *tuple* $>$
select $<$ *tuple* $>$	t2	
COMMIT	t3	
	t4	ROLLBACK

L'état est correct, mais la lecture de l'instant t2 correspond à une valeur qui ne sera jamais présente physiquement dans la base. La transaction A n'est pas reproductible.

Les problèmes potentiels

Les dépendances non valides dues à un échec (2)

Transaction A	temps	Transaction B
	t1	update < *tuple* >
update < *tuple* >	t2	
COMMIT	t3	
	t4	ROLLBACK

L'état est correct, mais l'écriture de l'instant t2 validée à l'instant t3 sera annulée à l'instant t4. L'effet de la transaction A est perdu.

Les problèmes potentiels

Les états incorrects

Transaction A	temps	Transaction B
select $<$ *tuple* $>$	t1	
	t2	select $<$ *tuple* $>$
update $<$ *tuple* $>$	t3	
COMMIT	t4	
	t5	update $<$ *tuple* $>$
	t6	COMMIT

À l'instant t4, l'état est correct. À l'instant t6, B pense que l'état est correct, mais la mise à jour à l'instant t5 se fait à partir de la valeur obtenue à l'instant t2 et non à partir de la nouvelle valeur. L'effet de la transaction A est perdu. L'état n'est donc pas *correct*.

La sérialisation : concepts

Definition

Une exécution concurrente d'un ensemble de transactions est sérialisable si et seulement si il existe un ordonnancement des transactions qui, pour tout état correct, donne après l'exécution séquentielle le même état correct comme résultat.

Propriété

Une exécution sérialisable transforme un état correct en un état correct.

Propriété

Les trois exemples précédent ne sont pas sérialisables.

La sérialisation

Une solution : Les verrous

Les problèmes sont d'une part dus à des lectures de valeurs non valides; d'autre part dus à des conflits d'écritures. Une solution consiste à *verrouiller* les objets (relation, tuples, attribut) pendant leurs utilisations par une transaction de sorte que les autres transactions ne puissent pas y accéder.
Cette solution abouti à deux types de verrous : les verrous exclusifs (X locks) utilisés pour les écritures, et les verrous partagés (S locks) utilisés pour les lectures.

La sérialisation

Le verrouillage strict à deux phases : La structure

À chaque verrou, on associe une structure à trois champs :

- l'état du verrou (L, S, X),
- D l'ensemble des transactions qui détiennent le verrou,
- F une file de demandes de transactions en attente du verrou.

La sérialisation

Le verrouillage strict à deux phases : L'algorithme

Tous les accès et ordres sont mis dans la file et le protocole gère les verrous en fonction du contenu de F.

$(L, \emptyset, F)$	$head(F) = lecture(T_i)$	$(S, \{T_i\}, F - \{lecture(T_i)\})$
(S, D, F)	$lecture(T_i) \in F$	$(S, D \cup \{T_i\}, F - \{lecture(T_i)\})$
$(L, \emptyset, F)$	$head(F) = ecriture(T_i)$	$(X, \{T_i\}, F - \{ecriture(T_i)\}$
$(S, \{T_i\}, F)$	$ecriture(T_i) \in F$	$(X, \{T_i\}, F - \{ecriture(T_i)\}$
(S, D, F)	$COMMIT(T_i) \in F$	$(S, D - \{T_i\}, F - \{COMMIT(T_i)\})$
(S, D, F)	$ROLLBACK(T_i) \in F$	$(S, D - \{T_i\}, F - \{ROLLBACK(T_i)\}$
$(S, \{T_i\}, F)$	$COMMIT(T_i) \in F$	$(L, \emptyset, F - \{COMMIT(T_i)\})$
$(S, \{T_i\}, F)$	$ROLLBACK(T_i) \in F$	$(L, \emptyset, F - \{ROLLBACK(T_i)\})$
$(X, \{T_i\}, F)$	$COMMIT(T_i) \in F$	$(L, \emptyset, F - \{COMMIT(T_i)\})$
$(X, \{T_i\}, F)$	$ROLLBACK(T_i) \in F$	$(L, \emptyset, F - \{ROLLBACK(T_i)\})$

La sérialisation

Théorème du verrouillage à deux phases

Propriété (Théorème du verrouillage à deux phases)

Avec le verrouillage strict à deux phases, toutes les exécutions concurrentes possibles sont sérialisables.

Propriété (Corollaire)

Avec le verrouillage strict à deux phases, toutes les exécutions concurrentes possibles sont correctes.

La sérialisation

Exemple sur le problème 1

Transaction A	temps	Transaction B
	t1	update $< tuple >$
select $< tuple >$	t2	
COMMIT	t3	
	t4	ROLLBACK

Transaction A	temps	Transaction B	Verrou($tuple$)
	t1.1	rq($update(tuple)$)	$(L, \emptyset, \{update(B)\})$
	t1.2	$update(tuple)$	$(X, \{B\}, \emptyset)$
rq($select(tuple)$)	t2.1		$(X, \{B\}, \{select(A)\})$
	t4.1	rq($ROLLBACK$)	$(X, \{B\}, \{select(A), ROLLBACK(B)\})$
	t4.2	$ROLLBACK$	$(L, \emptyset, \{select(A)\})$
$select(tuple)$	t2.2		$(S, \{A\}, \emptyset)$
rq($COMMIT$)	t3.1		$(S, \{A\}, \{ROLLBACK(A)\})$
$COMMIT$	t3.2		$(L, \emptyset, \emptyset)$

L'état est correct et les deux transitions se sont exécutées.

La sérialisation

Exemple sur le problème 2

Transaction A	temps	Transaction B
	t1	update $< tuple >$
update $< tuple >$	t2	
COMMIT	t3	
	t4	ROLLBACK

Transaction A	temps	Transaction B	Verrou($tuple$)
$select(tuple)$	t1		$(S, \{A\}, \emptyset)$
	t2	$select(tuple)$	$(S, \{A, B\}, \emptyset)$
rq($update(tuple)$)	t3.1		$(S, \{A, B\}, \{update(A)\})$
	t5.1	rq($update(tuple)$)	$(S, \{A, B\}, \{update(A), update(B)\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Les updates ne pourront jamais être sélectionnés. Les transactions ne peuvent pas terminées, c'est une situation d'interblocage.

Les blocages

Interblocage

Definition (interblocage)

Un ensemble de transactions est interbloqué lorsqu'il existe un cycle dans le graphe d'attente défini par :

- les arcs $(T_i, tuple_j)$ qui indique que T_i attend un verrou sur $tuple_j$,
- les arcs $(tuple_i, T_j)$ qui indique que $tuple_i$ est détenu par T_j,

Les blocages

Interblocage : exemple

L'exemple le plus classique est :

Transaction A	temps	Transaction B	Verrou($tuple1$)	Verrou($tuple2$)
$update(t1)$	t1		$(X, \{A\}, \emptyset)$	$(L, \emptyset, \emptyset)$
	t2	$update(t2)$	$(X, \{A\}, \emptyset)$	$(X, \{B\}, \emptyset)$
rq($update(t2)$)	t3.1		$(X, \{A\}, \emptyset)$	$(X, \{B\}, \{A\})$
	t5.1	rq($update(t1)$)	$(X, \{A\}, \{B\})$	$(X, \{B\}, \{A\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Les blocages

Attente infinie

Definition (Attente infinie)

Une attente infinie correspond à une exécution au cours de laquelle une transaction reste toujours dans la file d'attente.

Propriété

Avec le verrouillage strict à deux phases, les exécutions concurrentes ne sont pas exemptes d'interblocages et d'attentes infinies.

Les blocages

Attente infinie : exemple

Tr. A	temps	Tr. B	Tr. C	Verrou(t)
$select(t)$	t1			$(S, \{A\}, \emptyset)$
	t2	$update(t)$		$(S, \{A\}, \{B\})$
	t(3+4i)		$select(t)$	$(S, \{A, C\}, \{B\})$
$COMMIT$	t(4+4i)			$(S, \{C\}, \{B\})$
$select(t)$	t(5+4i)			$(S, \{A, C\}, \{B\})$
	t(6+4i)		$COMMIT$	$(S, \{A\}, \{B\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Des solutions aux problèmes du blocage

Détection et correction

Pour l'interblocage, il faut mettre en place un algorithme qui calcule le graphe d'attente et détecte la présence d'un cycle.
Lorsqu'un cycle est découvert, il faut choisir (arbitrairement) une transaction T_i, l'interrompre et effectuer $ROLLBACK(T_i)$.
Pour l'attente infinie, c'est plus compliqué.

Des solutions aux problèmes du blocage

Évitement

Une solution consiste à adapter le protocole à deux phases en mémorisant pour chaque transaction une estampille qui est sa date de création. Cette estampille sert pour soit tuer une transaction, soit s'auto-détruire. Deux versions lorsque (T_i, e_i) demande un verrou sur $tuple_j$ détenu par (T_k, e_k).

Wait-Die : si $e_i < e_k$, T_i attend, sinon T_i meurt.

Wound-Wait : si $e_i < e_k$, T_i blesse T_k, sinon T_i attend.

Dans les deux cas, la transaction tuée redémarre plus tard en gardant son estampille d'origine.

Des solutions aux problèmes du blocage

Évitement : exemple avec Wait-Die

Transaction A	temps	Transaction B	Verrou($t1$)	Verrou($t2$)
update($t1$)	t1		$(X, \{(A, t1)\}, \emptyset)$	$(L, \emptyset, \emptyset)$
	t2	*update*($t2$)	$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(B, t2)\}, \emptyset$
rq(*update*($t2$))	t3.1		$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(B, t2)\}, \{(A, t$
	t5.1	rq(*update*($t1$))	$(X, \{(A, t1)\}, \{(B, t2))$	$(X, \{(B, t2)\}, \{(A, t$
	t6	*ROLLBACK*	$(X, \{(A, t1)\}, \emptyset)$	$(L, \emptyset, \{(A, t1)\})$
update($t2$)	t3.2		$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(A, t1)\}, \emptyset$
COMMIT	t7		$(L, \emptyset, \emptyset)$	$(L, \emptyset, \emptyset)$
	t8	*update*($t2$)	$(L, \emptyset, \emptyset)$	$(X, \{(B, t2)\}, \emptyset$
	t9	*update*($t1$)	$(X, \{(B, t2)\}, \emptyset)$	$(X, \{(B, t2)\}, \emptyset$
	t10	*COMMIT*	$(L, \emptyset, \emptyset)$	$(L, \emptyset, \emptyset)$

Des solutions aux problèmes du blocage

Évitement : propriété

Propriété

Le protocole de verrouillage à deux phases avec évitement garanti que les exécutions concurrentes de transactions sont correctes et sans interblocage. Avec un seul type de verrou, il n'y a pas d'attente infinie. Avec deux types de verrou, l'attente infinie est possible.

Et en pratique

Du point de vue des SGBD

- Il y a peu de SGBD qui implémente le protocole de verrouillage à deux phases avec évitement.
- La plus part implémente le protocole de verrouillage à deux phases (strict ou une variante qui libère un peu plus tôt les verrous).
  - Peu implémente la détection de cycle dans le graphe d'attente.
  - La plus part estime qu'une transaction est bloquée (interblocage ou attente infinie) lorsqu'elle est depuis longtemps dans une file d'attente d'un verrou. Le SGBD la tue puis la relance.
- La plus part des SGBD trouve que la pose des verrous ralenti trop l'exploitation de la BD.
  - Ils distinguent différents niveau d'isolation des tuples. C'est une heuristique qui statistiquement fonctionne, mais qui peut-être risquée.

Et en pratique

Du point de vue SQL

- SQL n'impose pas la pose de verrou (d'autres techniques existent) mais exige le respect de la correction.
- La norme SQL propose plusieurs niveaux d'isolation qui peuvent être spécifiés en début de transaction : `SERIALIZABLE`, `REPEATABLE READ`, `READ COMMITTED` et `READ UNCOMMITTED`.
- PostGreSQL propose les 4 niveaux mais en implémente que deux : `SERIALIZABLE`, `READ COMMITTED`.

Et en pratique

Du point de vue exploitation

Une bonne solution souvent utilisée :

- Une base accessible en lecture uniquement. Les verrous sont inutiles.
- Une base pour les modifications. Le seul type X pour les verrous, et donc ni blocage, ni attente infinie.
- Une recopie de la base modifiée à intervalle de temps régulier, ou bien la nuit.

Outline

L'optimisation des Bases de Données

Les objectifs

- Faire en sorte qu'une requête s'exécute le plus rapidement possible.
- Les bases de données sont souvent très grandes et ne peuvent être en mémoire vive. Le critère d'optimisation est donc le nombre d'entrées-sorties sur disque, en terme de nombre de pages (unité logique d'accès au disque).

L'optimisation des Bases de Données

Les contraintes

- Les développeurs des applications ne connaissent ni les structures de stockages des données, ni les tailles des différentes relations.
- Les développeurs ne veulent pas modifier leurs applications si les structures des BD changent.

L'optimisation des Bases de Données

La solution

- L'optimisation, si elle est possible, ne peut être qu'au niveau du SGBD.
- L'optimisation doit être paramétrable, car les contraintes peuvent changer d'un BD à une autre.

Un exemple

Le schéma et les hypothèses sur les données

- Les tables de TDs :
  $S(S\#, SNAME, STATUS, CITY)$,
  $P(P\#, PNAME, COLOR, WEIGHT, CITY)$,
  $SP(S\#, P\#, QTY)$.
- 100 fournisseurs, 10000 commandes dont 50 pour la pièce $P2$.
- Chaque tuple utilise une page en mémoire.

Un exemple

Une requête

La requête *Obtenir les noms des fournisseurs de la pièce P2*.

$$Req = \pi[SNAME](\sigma[P\# = P2](S \bowtie SP))$$

La complexité du calcul

1. La jointure $S \bowtie SP$ coûte 10000 x 100 opérations et fabrique une relation ayant 10000 tuples.
2. La sélection $\sigma[P\# = P2]$ coûte 10000 opérations et fabrique une relation ayant 50 tuples.
3. La projection $\pi[SNAME]$coûte 50 opérations.

Soit un total de 1020100 opérations d'entrées-sorties.

Un exemple

Une requête équivalente

Soit $Req' = \pi[SNAME](\sigma[P\# = P2](SP) \bowtie S)$.
Nous avons $Req \equiv Req'$.

1. La sélection $\sigma[P\# = P2](\text{SP})$ coûte 10000 opérations et fabrique une relation TMP ayant 50 tuples.
2. La jointure $TMP \bowtie S$ coûte 50 x 100 opérations et fabrique une relation ayant 50 tuples.
3. La projection $\pi[SNAME]$coûte 50 opérations.

Soit un total de 15050 opérations d'entrées-sorties.

Un exemple

Une autre optimisation

Supposons que la relation SP possède des adressages indexés ou dispersés sur $P\#$ et sur $S\#$.
Avec $Req' = \pi[SNAME](\sigma[P\# = P2](SP) \bowtie S)$.

1. La sélection $\sigma[P\# = P2](\mathrm{SP})$ coûte 50 opérations et fabrique une relation TMP ayant 50 tuples.
2. La jointure $TMP \bowtie S$ coûte 50 opérations et fabrique une relation ayant 50 tuples.
3. La projection $\pi[SNAME]$coûte 50 opérations.

Soit un total de 150 opérations d'entrées-sorties.

Un exemple

Idées pour l'optimisation

Schématiquement, l'optimisation d'une BD dans un SGBD consiste à utiliser les techniques suivantes :

- Réécriture automatiquement des requêtes.
- Calculs statistiques sur les relations pour utiliser les bonnes heuristiques.
- Ajouts de structures (décision de l'administrateur) pour accéder rapidement aux informations les plus fréquemment demandées.

La réécriture de requêtes

Propriétés

- Elle est indépendante des données.
- Elle fourni toujours une requête équivalente.
- La nouvelle requête est plus efficace en nombre d'entrées-sorties.

La réécriture de requêtes

Les opérateurs commutatifs et associatifs

$\forall$ op $\in \{\cup, \cap, \times, \bowtie, \theta[p]\}$, nous avons :

- R^1 op $R^2 = R^2$ op R^1
- R^1 op $(R^2$ op $R^3) = (R^1$ op $R^2)$ op R^3.

La réécriture de requêtes

Sélection et projection

- $\pi[A_i^1](\pi[A_i^2](\ldots(\pi[A_i^k](R)))) = \pi[\cap_{j=1}^{j=k} A_i^j](R)$
- $\sigma[P^1](\sigma[P^2](\ldots(\sigma[P^k](R)))) = \sigma[\bigwedge_{j=1}^{j=k} P^j](R)$
- $\pi[A_i](\sigma[P^1](R)) = \sigma[P^1](\pi[A_i](R))$

La réécriture de requêtes

Idempotence et absorption

- $\forall$ op $\in \{\cup, \cap, \bowtie\}$, nous avons R op $R = R$
- $R^1 \cup (R^1 \cap R^2) = R^1$
- $R^1 \cap (R^1 \cup R^2) = R^1$

La réécriture de requêtes

Forme Normale Conjonctive (CNF)

Les expressions booléennes sont mises en CNF.

- $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

La réécriture de requêtes

Les transformations sémantiques

Elles utilisent les dépendances fonctionnelles pour réécrire les requêtes.

- $\pi[P\#](SP \bowtie S) = \pi[P\#](SP)$ du fait que c'est la jointure sur $S\#$ qui est une clef candidate de S et une clef étrangère de SP.

Ces transformations peuvent être très efficaces, mais malheureusement presque jamais implémentées.

La réécriture de requêtes

Diviser et conquérir

Soit le problème :

- *Les noms des fournisseurs londoniens ayant une commande d'une pièce rouge pesant moins de 25 kilos avec une quantité supérieure à 200.*
- $\pi[SNAME](\sigma[S.CITY = Londres \ \wedge \ SP.QTY > 200 \ \wedge \ P.COLOR = Red \ \wedge \ P.WEIGHT < 25](S \bowtie SP \bowtie P))$

Une technique efficace :

1. Paralléliser des sous tâches indépendantes pour créer des tables temporaires :
   - $P' = \sigma[COLOR = Red \ \wedge \ WEIGHT < 25](P)$
   - $S' = \sigma[CITY = Londres](S)$
   - $SP' = \sigma[QTY > 200](SP)$
2. $\pi[SNAME](S' \bowtie SP' \bowtie P')$

Principe des heuristiques

En fonction :

- de la requête,
- de statistiques sur les données,
- des structures complémentaires présentes,

un SGBD va :

1. estimer les coûts de différentes solutions,
2. choisir celle qui semble la plus efficace,

et éventuellement mémoriser ce choix pour une utilisation ultérieure.

Exemple

Soient deux relations $R(A, C)$ et $S(B, C)$ et une requête nécessitant le calcul de $R \bowtie S$.

On suppose également :

	# tuples	# tuples par page	# pages	# valeurs C
R	tR=100	tpR=1	pR=100	nC=20
S	tS=10000	tpS=10	pS=1000	nC=50

L'estimation du nombre de tuples dans $R \bowtie S$ est :

$tRS = (100x10000)/max(20, 50) = 20000.$

Algorithmes et structures de données

L'algorithme naïf

```
Pour i de 1 a tR faire { // iteration externe tuples de R
  Pour j de 1 à tS faire { // iteration interne tuples de S
    si (R[i].C = S[j].C) alors
      ajouter (R[i].A, S[j]) au résultat.
```

La complexité en nombre de pages accédées est (externe + interne) :

- si $R \bowtie S$: $(tR/tpR) + tR * (tS/tpS) = 100100$ lectures et $tRS/tpRS$ écritures.
- si $S \bowtie R$: $(tS/tpS) + tS * (tR/tpR) = 1001000$ lectures et $tRS/tpRS$ écritures.

$\Rightarrow$ Il faut faire l'itération interne sur la relation ayant le plus de tuple par page.

Algorithmes et structures de données

L'algorithme avec recherche indexée (1)

La construction d'un index pour les tuples de S.

```
Pour j de 1 a tS faire {
  ind := S[j].C;
  ajouter ind dans le B-arbre
  // structure fréquemment utilisée pour les index
}
```

Le calcul de la jointure

```
Pour i de 1 a tR faire { // iteration externe tuples de R
  // X[1]...X[k] les entrées dans S pour l'index R[i].C
  Pour j de 1 à k faire { // iteration interne tuples de S
    ajouter (R[i].A, S[X[j]]) au résultat.
```

L'algorithme avec recherche indexée (2)

La complexité en nombre de pages accédées est (externe + index + interne) :

- si $R \bowtie S$: $(tR/tpR) + tR * (h = 3) + tRS/tpS = 2400$ lectures et $tRS/tpRS$ écritures.
- si $S \bowtie R$: $(tS/tpS) + tS * (h = 2) + tRS/tpR = 41000$ lectures et $tRS/tpRS$ écritures.

où h est la hauteur du B-arbre.
$\Rightarrow$ Il faut faire l'itération interne sur la relation ayant le plus de tuple par page.

L'algorithme avec recherche dispersée (1)

La construction d'un adressage dispersé pour S.

```
Pour j de 1 a tS faire {
  ind := hash(S[j].C);
  ajouter S[j] à l'entrée H[ind]
}
```

Le calcul de la jointure

```
Pour i de 1 a tR faire { // iteration externe tuples de R
  ind := hash(R[i].C);
  // S[1]...S[k] les tuples de S à l'adresse H[ind]
  Pour j de 1 à k faire // iteration interne tuples de S
    // il faut tester pour les éventuels conflits
    si (R[i].C = S[j].C) alors
      ajouter (R[i].A, S[j]) au résultat.
}
```

L'algorithme avec recherche dispersée (2)

La complexité en nombre de pages accédées est (externe + conflit + interne) :

- si $R \bowtie S$: $(tR/tpR) + (\alpha = 1,5)(tRS/tpS) = 3100$ lectures et $tRS/tpRS$ écritures.
- si $S \bowtie R$: $(tS/tpS) + (\alpha = 1,5)(tRS/tpR) = 31000$ lectures et $tRS/tpRS$ écritures.

$\Rightarrow$ Il faut faire l'itération interne sur la relation ayant le plus de tuple par page.

L'algorithme tri+fusion (1)

Supposons R et S triées sur l'attribut C (sinon tri avant).

```
r := 1; s := 1;
tant que (r <= tR & s <= tS) faire {
  j := s;
  tant que (j <= tS && S[j].C < R[r].C) faire j++;
  tant que (j <= tS && S[j].C = R[r].C) faire {
    i := r
    tant que (i <= tR && R[i].C = R[r].C) faire {
      ajouter (R[i].A, S[j]) au résultat.
      i++
    }
    j++;
  }
  r := i; s := j;
}
```

L'algorithme tri+fusion (2)

Les seuls tuples lus plusieurs fois sont dans la boucle la plus interne. Il est raisonnable de penser qu'ils tiennent ensemble en mémoire vive, de sorte qu'ils ne génèrent qu'une entrée-sortie. La complexité en nombre de pages accédées est (externe + interne) :

- si $R \bowtie S$: $(tR/tpR) + (tS/tpS) = 1100$ lectures et $tRS/tpRS$ écritures.
- si $S \bowtie R$: $(tS/tpS) + (tR/tpR) = 1100$ lectures et $tRS/tpRS$ écritures.

C'est l'algorithme le plus efficace si les relations sont triées par rapport à C.

Bilan sur l'optimisation

- L'optimisation est une tâche complexe.
- Pour de grandes bases de données, les administrateurs doivent être compétents.
- Beaucoup d'heuristiques (dé-normalisation, ...)

Outline

Conception des Bases de Données

Schéma relationnel

- un ensemble de schémas de relations $\{Schema(R^i) =< (A^i_1, A^i_2, \ldots, A^i_{n_i}), \{I^i\}, P^i >\}$,
- pour chaque relation R^i, un ensemble (éventuellement vide) d'identifiants externes,
- une contrainte d'intégrité inter-relations (éventuellement *True*).

Conception des Bases de Données

Les niveaux d'abstractions

Externe : Les vues.

Conceptuel : Les attributs.

Physique : Les types de données.

Les applications

- Les requêtes.
- Les langages avec requêtes.

Conception des Bases de Données

L'optimisation

- Exploitation : Verrous, BD écriture et BD lecture ?
- Les index et autres structures.
- La dénormalisation.
- Les procédures stockées.

Conception des Bases de Données

Méthodologie de conception

1. Lecture attentive du cahier des charges. Attention, il y a beaucoup de “non-dits”, relatifs au domaine de l’application.
2. Identification des attributs.
3. Identification des dépendances fonctionnelles : celles du cahier des charges; les “non-dites” (à décrire) et les supplémentaires éventuelles (à justifier).
4. Identification des contraintes d’intégrité : celles du cahier des charges; les “non-dites” (à décrire) et les supplémentaires éventuelles (à justifier)..
5. Couche physique : typage des attributs.
6. Normalisation : BCNF ou 3NF.
7. Identification des fonctionnalités : les vues et les requêtes.
8. Optimisation de la BD.

Méthodologie de conception : un exemple

Le cahier des charges : un dialogue (début)

Réceptionniste: Hôtel du Nord, Bonjour !
Client: Bonjour, ce serait pour réserver une chambre du 1^{er} au 4 juillet dans votre hôtel. Quels sont vos tarifs?
R. Nos prix s'entendent petit déjeuner compris. La Single est à 47euros, la Double 65euros et la Workable 92euros, c'est une double avec un bureau tout équipé et accès illimité à Internet haut débit. Quelle catégorie vous intéresse ?
C. Je souhaite une Double ordinaire.
R. Je vérifie la disponibilité? oui, il nous reste des doubles au 1^{er} et au 4^{e} étage. Que préférez-vous?
C. Euh? au 1^{er}.
R. Très bien. Puis-je vous demander votre nom ?
h Albert Duchemin.

...

Méthodologie de conception : un exemple

Le cahier des charges : un dialogue (fin)

R. ... Je vois que vous êtes un client fidèle!
C. Ah bon ?
R. Oui vous habitez bien 10 rue des Jumeaux à Strasbourg?
C. Ah, non ! ça, c'est mon grand-oncle...
R. Je dois donc vous enregistrer. Pouvez-vous me donner votre adresse ?
C. 13, rue de la Paix à Saillans, code postal 33141.
R. Voilà ! Tiens, vous êtes notre dix millième client ! On attribue un numéro différent à chaque client et vous êtes le numéro 10000 !
C. Qu'est ce que j'ai gagné ?
R. On fera sans doute quelque chose pour votre arrivée ! Bien, je récapitule, j'ai réservé, au nom de M. Albert Duchemin résidant 13 rue de la Paix, à Saillans, une chambre de catégorie double, la 12^e du 1^{er} étage pour être précis, et du 1^{er} juillet au 4 juillet matin.
C. Très bien ! Je vous remercie. Au revoir.
R. Je vous en prie, au revoir M. Duchemin.

Méthodologie de conception : un exemple

Le cahier des charges : annexe

De plus, notons qu'un client peut réserver plusieurs chambres en même temps ou pas. Une chambre peut être réservée par au plus un client pour un jour donné. Les étages sont numérotés à partir de 1 et pour un étage donné, les chambres sont numérotées à partir de 1. Enfin les consommations d'un client sont enregistrées quotidiennement: chaque consommation est désignée par un intitulé qui lui est propre et possède un prix unitaire. Par exemple, voici les consommations de M. Duchemin du 1er juillet 2010:

Intitulé	Quantité	Prix Unitaire	Montant
Pepsi-Cola	4	2	8
Téléphone	12	0,4	4,8
Whisky - baby	1	9	9
Sauna	1	11	11
		Total du 1/7/2010	32,8 euros

Méthodologie de conception : un exemple

Lecture du cahier des charges : les attributs

- DateDebut : Date début réservation d'une chambre.
- DateFin : Date fin réservation d'une chambre.
- Tarif : Tarifs d'une chambre.
- Categorie : Catégorie d'une chambre.
- Descriptif : Descriptif des services d'une chambre.
- Disponibilite : Disponibilité d'une chambre.
- Etage : L'étage d'une chambre.
- Nom, Prenom : Les noms et prénoms des clients.
- Fidele : La fidélité d'un client.
- Adresse : L'adresse d'un client.
- NumClient : Le numéro d'ordre d'un client depuis le premier.
- Numero : Le numéro de la chambre à l'étage.
- Intitule : L'intitulé d'une boisson.
- PrixUnitaire : Le prix d'une boisson.
- Quantite : Le nombre de consommation par Intitulé, par jour et par client.

Méthodologie de conception : un exemple

Lecture du cahier des charges : les contraintes

- Une chambre a au plus un client par jour.
- Les étages sont numérotés depuis 1.
- A chaque étage, les chambres sont numérotés depuis 1.
- Les homonymes sont possibles (mais pas à la meme adresse).
- Un client peut réserver simultanément plusieurs chambres.

Lecture du cahier des charges : les fonctionnalités

- Disponibilite : Disponibilité d'une chambre.
- Fidele (Attribut/Fonctionnalite) : La fidélité d'un client.
- Alerte (Un client paut gagner).
- Facture (Par chambre ?, par client ?, par jour ?).
- Chambre en cours de rénovation (attribut ?, type de chambre ?).

Méthodologie de conception : un exemple

Lecture du cahier des charges : les décisions

- Disponibilite (Fonctionnalite) : Disponibilité (pour une date ?, pour plusieurs jours consécutifs ?, avec changement de chambre ?). Le receptionniste proposera.
- Fidelite (Fonctionnalite) : La fidélité d'un client est le résultat d'un calcul et non au bon vouloir du réceptionniste.
- Facture : à la date de fin de chaque réservation, avec cumul des consommations du client. $\rightarrow$ un attribut "Règlée" ?
- "Rénovation en cours" est un type de chambre.

Méthodologie de conception : un exemple

Identification des dépendances fonctionnelles

- $Categorie \rightarrow (Tarif, Descriptif)$
- $(Etage, Numero) \rightarrow Categorie$
- $(Jour, NumClient, Intitule) \rightarrow Quantite$
  Il faut ajouter l'attribut Jour. (Entre les dates de début et de fin ?)
- $Intitule \rightarrow PrixUnitaire$
- $NumClient \rightarrow (Nom, Prenom, Adresse)$
- $(Nom, Prenom, Adresse) \rightarrow NumClient$
- $(DateDebut, Etage, Numero) \rightarrow (DateFin, NumClient)$
- $(DateFin, Etage, Numero) \rightarrow (DateDebut, NumClient)$
  Une contrainte entre les dates de début et de fin.

Méthodologie de conception : un exemple

Identification des contraintes d'intégrité

Des contraintes élémentaires.

- DateFin d'une réservation est postérieure à DateDebut.

Des contraintes statiques

- Les numéros des clients sont séquentiels à partir de 1.
- (Etage, Numero) doivent etre séquentiels.
- (Etage, Numero, DateDebut, DateFin) ne doivent pas "se chevauchées".

Méthodologie de conception : un exemple

Couche physique : Typage des attributs

text	Categorie, Descriptif, Intitule, Nom, Prenom, Adresse.
date	DateDebut, DateFin, Jour.
serial	NumClient (traite automatiquement la contrainte).
integer > 0	Tarif, Etage, Numero, Quantite.
real > 0	PrixUnitaire.

Méthodologie de conception : un exemple

Normalisation

1. Algorithme DFs $\rightarrow$ BCNF.
2. si une DF devient une C.I.
   - 2.1 Algorithme DFs $\rightarrow$ 3NF.
   - 2.2 Choisir entre BCNF et 3NF au vu des résultats.

Méthodologie de conception : un exemple

Normalisation BCNF

Deux clefs candidates

1. (Etage, Numero, DateDebut, Jour, Intitule)
2. (Etage, Numero, DateFin, Jour, Intitule)

Une décomposition parmi les possibles

1. Categorie $\neq C_i \Rightarrow$ TypeChambre=(Categorie, Tarif, Descriptif)
2. (Etage, Numero) $\neq C_i \Rightarrow$ Chambres=(Etage, Numero, Categorie)
3. Intitule $\neq C_i \Rightarrow$ PrixConsommation=(Intitule, PrixUnitaire)
4. NumClient $\neq C_i \Rightarrow$ Clients=(NumClient, Nom, Prenom, Adresse)
5. (NumClient, Jour, Intitule) $\neq C_i \Rightarrow$ Consommations=(NumClient, Jour, Intitule, Quantite)
6. Reservations=(Etage, Numero, DateDebut, DateFin, NumClient) est BCNF.

Il n'y a pas de DF qui porte sur plusieurs relations $\rightarrow$, on garde cette décomposition en BCNF.

Méthodologie de conception : un exemple

La relation TypeChambre

```
CREATE TABLE Hotel.TypeChambre (
  -- NOT NULL pour etre relationnel
   Categorie text NOT NULL,
   Tarif integer NOT NULL CHECK (Tarif > 0),
   Descriptif text NOT NULL,
  -- Clefs candidates
   PRIMARY KEY (Categorie)
);
```

Méthodologie de conception : un exemple

La relation Chambres

```
CREATE TABLE Hotel.Chambres (
  -- NOT NULL pour etre relationnel
  Etage integer NOT NULL CHECK(Etage > 0),
  Numero integer NOT NULL CHECK(Numero > 0),
  Categorie text NOT NULL,
  -- Clefs candidates
  PRIMARY KEY (Etage, Numero),
  -- Clefs Ã©trangÃ¨res
  FOREIGN KEY (Categorie) REFERENCES Hotel.TypeChambre(C
  -- Contrainte integrite statique
  -- coherence (etage,numero) definie par une fonction
);
```

Méthodologie de conception : un exemple

Une C.I. statique pour Chambres

```
-- PostgreSQL n'accepte pas les contraintes d'integrite
-- Il faut definir une fonction
-- L'unicite n'est pas testÃ©e ici car prÃ©sente dans la
CREATE FUNCTION Hotel.EtageNumeroCorrects(integer, integ
  RETURNS boolean AS $$
SELECT ($1 = 1 AND $2 = 1)
  OR EXISTS (
  SELECT *
  FROM Hotel.Chambres
  WHERE ($1 = Etage AND $2 = (Numero + 1))
     OR ($1 = (Etage + 1) AND $2 = 1))
$$ LANGUAGE SQL;
-- puis appeler la fonction dans une contrainte d'integr
ALTER TABLE Hotel.Chambres ADD CONSTRAINT NumeroPossible
  CHECK(Hotel.EtageNumeroCorrects(Etage, Numero) = TRUE)
```

Méthodologie de conception : un exemple

La relation PrixConsommation

```
CREATE TABLE Hotel.PrixConsommation (
  -- NOT NULL pour etre relationnel
   Intitule text NOT NULL,
   PrixUnitaire real NOT NULL CHECK(PrixUnitaire > 0),
  -- Clefs candidates
  PRIMARY KEY (Intitule)
);
```

Méthodologie de conception : un exemple

La relation Clients

```
CREATE TABLE Hotel.Clients (
  -- NOT NULL pour etre relationnel
  NumClient serial NOT NULL,
  Nom text NOT NULL,
  Prenom text NOT NULL,
  Adresse text NOT NULL,
  -- Clefs candidates
  PRIMARY KEY (NumClient),
  UNIQUE (Nom, Prenom, Adresse)
);
```

Méthodologie de conception : un exemple

La relation Consommations

```
CREATE TABLE Hotel.Consommations (
  -- NOT NULL pour etre relationnel
   NumClient serial NOT NULL,
   Jour date NOT NULL,
   Intitule text NOT NULL,
   Quantite integer NOT NULL CHECK (Quantite > 0),
   -- Clefs candidates
   PRIMARY KEY (NumClient, Jour, Intitule),
   -- Clefs Ã©trangÃ¨res
   FOREIGN KEY (NumClient) REFERENCES Hotel.Clients(NumC
   FOREIGN KEY (Intitule) REFERENCES Hotel.PrixConsommati
);
```

Méthodologie de conception : un exemple

La relation Reservations

```
CREATE TABLE Hotel.Reservations (
  -- NOT NULL pour etre relationnel
  Etage integer NOT NULL,
  Numero integer NOT NULL,
  DateDebut date NOT NULL,
  DateFin date NOT NULL,
  NumClient serial NOT NULL,
  -- Clefs candidates
  PRIMARY KEY (Etage, Numero, DateDebut),
  UNIQUE (Etage, Numero, DateFin),
  -- Clefs externes
  FOREIGN KEY (NumClient) REFERENCES Hotel.Clients(NumC
  FOREIGN KEY (Etage, Numero) REFERENCES Hotel.Chambres(
  -- Contrainte intÃ©gritÃ© Ã©lÃ©mentaire
  CHECK (DateDebut < DateFin)
  -- Contrainte intÃ©gritÃ© statique
  -- cohÃ©rence (Etage, Numero, DateDebut, DateFin) defi
)
```

Méthodologie de conception : un exemple

Une C.I. statique pour Reservations

```
-- PostgreSQL n'accepte pas les contraintes d'integrite
-- Il faut definir une fonction
CREATE FUNCTION Hotel.DatesDebutFinCorrectes(integer, in
  RETURNS boolean AS $$
SELECT NOT EXISTS (
  SELECT *
  FROM Hotel.Reservations
  WHERE $1 = Etage
    AND $2 = Numero
    AND $3 <= DateFin
    AND $4 >= DateDebut)
$$ LANGUAGE SQL;
-- puis appeler la fonction dans une contrainte d'integr
ALTER TABLE Hotel.Reservations ADD CONSTRAINT DatesPossi
  CHECK(Hotel.DatesDebutFinCorrectes(Etage, Numero, Date
```

Méthodologie de conception : un exemple

Fonctionnalité : l'alerte

-- *les alertes pour des situations exceptionnelles*
-- *Le numero du client est un multiple de 500*
-- *ou bien il arrive le jour de l'anniversaire du patron*

```
CREATE VIEW Hotel.Alerte AS
SELECT Nom, Prenom, Adresse
FROM   Hotel.Clients
WHERE  NumClient % 500 = 0
UNION
SELECT Nom, Prenom, Adresse
FROM   Hotel.Clients NATURAL JOIN Hotel.Reservations
WHERE  extract(month from DateDebut)=7
  AND  extract(day from DateDebut)=21;
```

Méthodologie de conception : un exemple

Fonctionnalité : la fidélité

```
-- les clients fidÃ¨les
-- Ils sont venus plus de 10 fois
-- ou bien ils ont reservÃ©s plus de 3 fois une chambre
```

```
CREATE VIEW Hotel.Fidelite AS
SELECT Nom, Prenom, Adresse
FROM   Hotel.Clients NATURAL JOIN Hotel.Reservations
GROUP BY NumClient, Nom, Prenom, Adresse
HAVING COUNT(*)>10
UNION
SELECT Nom, Prenom, Adresse
FROM   Hotel.Clients NATURAL JOIN Hotel.Reservations NAT
WHERE  Categorie = 'Workable'
GROUP BY NumClient, Nom, Prenom, Adresse, Categorie
HAVING COUNT(*)>3;
```

Méthodologie de conception : un exemple

Fonctionnalité : la disponibilite

```
-- Pour calculer la disponibilite
-- 1/ les debuts d'occupation de chaque chambre
--     un dÃ©but virtuel Ã  la fin du monde
-- 2/ les fins d'occupation de chaque chambre
--     une fin virtuelle Ã  la crÃ©ation du monde
-- 3/ les pÃ©riodes non occupÃ©es ce chaque chambre Ã  p
-- 4/ la meme chose avec de l'information complÃ©mentair

CREATE VIEW Hotel.Disponibilite AS
SELECT Categorie, Etage, LibreDu, Au, Numero, Descriptif
FROM (SELECT Etage, Numero, GREATEST(DateFin, current_da
      FROM   (SELECT Etage, Numero, DateDebut
              FROM   Hotel.Reservations
                UNION
              SELECT Etage, Numero, 'infinity'
              FROM   Hotel.Chambres) AS DebutOccupation
         NATURAL JOIN
```

Méthodologie de conception : un exemple

Optimisation de la BD

- Exploitation : Verrous, BD écriture et BD lecture ?
- Les index et autres structures.
- La dénormalisation.
- Les procédures stockées.